Multi-Core Devices for Safety-Critical Systems: A Survey

Jon Perez Cerrolaza[†], Roman Obermaisser[‡], Jaume Abella[⋆], Francisco J. Cazorla[⋆],
Kim Grüttner[§], Irune Agirre[†], Hamidreza Ahmadian[‡], Imanol Allende[†]
[†]Ikerlan
[‡]Universität Siegen
[⋆]Barcelona Supercomputing Center (BSC)
[§]OFFIS – Institute for Information Technology

November 4, 2020

Abstract

Multi-core devices are envisioned to support the development of next-generation safety-critical systems, enabling the on-chip integration of functions of different criticality. This integration provides multiple system-level potential benefits such as cost, size, power, and weight reduction. However, safety certification becomes a challenge and several fundamental safety technical requirements must be addressed, such as temporal and spatial independence, reliability, and diagnostic coverage. This survey provides a categorization and overview at different device abstraction levels (nanoscale, component, and device) of selected key research contributions that support the compliance with these fundamental safety requirements.

1 Introduction

Over the past decades, embedded systems have enabled tremendous improvements with respect to functionality, dependability and performance in many application domains such as transportation and industrial control systems. In these domains, the embedded systems often play a crucial role in guaranteeing the overall safety of the system. These systems are referred to as *safety-critical systems* as their failure can derive into catastrophic consequences such as the loss of lives or severe environmental damage [1] (e.g., automotive cruise control [2], railway signalling [3], wind turbine integrity protection [4], pacemaker [5]). With the aim of reducing the risk of causing such fatalities, safety-critical systems must follow strict certification processes according to domain specific safety standards. This process usually involves high development efforts and costs. As a general rule, the higher the safety integrity level, the higher the cost of safety certification [6, 7].

In addition, with the increasing digitization trend, where an increasing number of functionalities are implemented by software, the embedded systems often include functions with different safety criticalities that must also co-exist with non-critical software, conforming *mixed-criticality* systems. In the past, the mixed-criticality architecture has generally followed a federated architecture approach in which each major functionality is deployed on a dedicated computing node. The growing demand for additional functionality, leads to an increase in the number of computing nodes, wires and connectors. As a consequence, this leads to an increase in the overall cost, complexity, Size, Weight, and Power (SWaP) that in several cases limits the future scalability of this approach [7, 4, 6, 8, 9, 10]. For example, in the automotive domain, premium cars have more than 20 million lines of code deployed in around 100 computing nodes [11, 8] and electronic components added value ranges between 40% for traditional vehicles up to 75% for electric vehicles [12]. The current automotive evolution targets the development of intelligent Advanced Driver-Assistance Systems (ADAS) and autonomous driving solutions that will further increase the number of functionalities to be integrated [8].

One possible solution is the shift towards an integrated architectural approach [7, 4, 6, 8, 9, 10], where functions of different safety criticality are integrated in a reduced number of centralized computing nodes and processing devices. In this approach, safety certification becomes a challenge because the integration of mixed-criticality functions requires justifying sufficient independence of implementation and sufficiently low probability of dependent failure between functions [13, 14, 6, 7, 15]. Moreover, this approach requires an increase in the overall computational performance of devices, which could potentially be achieved by means of multi-core devices and mono-core devices with higher frequency. The usage of mono-core devices with higher frequency is considered not competitive in several domains due to increased sensibility to Electromagnetic Interference (EMI) [16], low reliability of thermal dissipation fans [8, 4] and cooling systems volume and weight [17]. On the other hand, Commercial Off-The-Shelf (COTS) multi-core devices are becoming dominant in silicon manufacturer roadmaps [18, 19, 20, 12, 21, 22, 10] and provide a cross-domain potential solution, e.g., automotive [23, 15, 2], avionics [24, 10], railway [3, 25], industrial control [6, 26], medical applications [5].

In this scenario, safety-critical system developers of multi-core device based mixed-criticality systems, need to comply with two sometimes conflicting and contradictory constraints. On the one hand, conservative functional safety standards based on the best safety industrial practices of the last decades, with none or limited consideration of multi-core devices (see

advances in ISO 26262-11 [14]). On the other hand, a rapidly evolving and highly innovative semiconductor industry that produces multi-core devices with shrinking technologies and higher integration of cores [27].

This publication is a survey on multi-core devices for safety-critical systems. The bulk of the research in multi-core devices for dependable systems focuses on specific challenges such as device and component architectures, reliability, time predictability and safety certification strategies [7, 5]. This survey however, aims to provide a categorization and end-to-end view of safety implications on multi-core devices, starting from nanoscale level and up to device level, by bringing together and summarizing the most relevant research contributions that support the compliance with fundamental safety technical requirements. This survey is aimed at researchers, with the aim of offering a panoramic view (survey of surveys) of the set of fundamental technical challenges that need to be addressed at different abstraction levels. This survey is also aimed at multi-core device and safety system developers. It provides an updated categorized research state-of-the-art, which provides a starting point to go through the (complex) design decision process required in the development of safety-critical systems, as required by diligence and liability requirements when the applicable safety standard has none or limited consideration of multi-core devices [28].

The reminder of the survey is organized as follows. Section 2 summarizes basic concepts and terminology. Section 3 analyzes, categorizes and summarizes the state-of-the-art with selected key research contributions that support the compliance with fundamental safety technical requirements at different device abstraction levels (nanoscale, component and device). Then, Section 4 summarizes avionics and space domain specific safety technical considerations, and summarizes relevant example case-studies from several domains. Finally, Section 5 provides links to related research topics, overall conclusion and future research directions.

2 Basic Concepts and Terminology

This survey uses the basic concepts, taxonomy and terminology defined by Avižienis et al. [1] for dependable and secure computing. In addition to this, this survey integrates various research fields with specific detailed concepts and terminology described by referenced major survey publications of each specific research field (e.g., nanoscale multi-core device dependability [5]).

The term multi-core device, or device for short, is used within this survey to refer to multi-core processors, System-on-a-Chip (SoC), Multi-Processor System-on-a-Chip (MPSoCs), FPGA with soft-cores and combinations of the previous. And the term computing node is used within this survey to refer to domain specific terms such as Electronic Control Unit (ECU) for automotive and Line Replaceable Unit (LRU) for avionics.

2.1 Safety certification and standards

Safety is defined as the "absence of catastrophic consequences on the user(s) and the environment" [1]. In several cases, safety-critical systems must be approved by an independent certification agency as part of a certification process. IEC 61508 [13] is a generic international safety standard considered a reference for other domain specific standards such as automotive (ISO 26262 [14]) and railway (EN 5012X [29]). For further details see the survey [30] that describes safety (and security) qualification and certification state-of-the-art and challenges for automotive, railway, space and avionics domains. Taking into consideration the variability of domain specific terms and requirements, this survey uses IEC 61508 as the reference safety standard. Section 4 provides a simplified survey of solutions specific for other additional domains such as avionics and space.

In IEC 61508, "Safety Integrity Level (SIL) is a discrete level corresponding to a range of safety integrity values where 4 is the highest and 1 is the lowest" [4]. The most stringent safety systems such as railway signaling systems (SIL4) are implemented with programmable electronics with a probability of dangerous failure in the range of 10^{-9} hours of operation ($\approx$ 114.155 years). This requires stringent technical solutions to reduce to such low level the probability of catastrophe due to systematic faults (e.g., method to reduce the probability of human, process and tool errors) and random faults (e.g., safety mechanism with a given Diagnostic Coverage (DC)).

A safety function is either fail-safe, if a safe state can be reached either by the safety function or by the diagnosis reaction (e.g., wind turbine stop [4], train stop [3]), or fail-operational if the system must guarantee full or degraded operation of the given function even in the presence of a failure (e.g., automotive autonomous driving system, avionics flight control system).

2.2 Fundamental safety technical requirements

The development of safety-critical systems that must comply with safety standards such as IEC 61508, requires an appropriate safety methodology to mitigate systematic errors and compliance with at least two fundamental safety technical requirements (reliability and diagnostic coverage (DC)). The development of safety-critical systems based on multi-core devices extends the previous with spatial and temporal independence requirements [13, 14] (see IEC 61508-3 Annex F). Thus, compliance with at least these four fundamental safety technical requirements is required.

- The *reliability* of a device at time t is the probability that the device will operate correctly during the period of time $[0, t]$. Reliability is measured as the failure rate of the device (λ) and expressed in Failures In Time (FIT), the number of failures expected in 10^9 hours of operation. For a given error, the effect can either be safe (not dangerous) or dangerous with respect to the associated safety function. Thus, as described in Equation 1, device failure rate is made up of the addition of components safe failure rates (λ_s) and dangerous failure rates (λ_d). The reliability of a device can be improved by techniques such as fault-tolerance.

- *Diagnostic Coverage (DC)* denotes the effectiveness of diagnosis techniques to detect dangerous errors, expressed in coverage percentage with respect to all possible dangerous errors. The DC is classified as low ($60\% - < 90\%$), medium ($90\% - < 99\%$) and high ($>= 99\%$) [13]. As shown in Equation 2, DC is calculated as the ratio between dangerous detected failure rate (λ_{dd}) and dangerous failure rate (λ_d), which includes both dangerous detected (λ_{dd}) and dangerous undetected failure rates (λ_{du}). The diagnostic coverage of digital circuits can be calculated and measured using methods such as Failure Mode and Effects Analysis (FMEA) and fault injection [31]. Dangerous detected errors usually require the definition of an associated reaction (e.g., activate safe state, error correction).

- *Temporal independence* is required to ensure that "one element shall not cause another element to function incorrectly by taking too high a share of the available processor execution time, or by blocking execution of the other element by locking a shared resource of some kind" [13], so "that elements will not adversely interfere with each other's execution behaviour such that a dangerous failure would occur" in the time domain [13]. For this purpose, as recommended by the standard [13, 14], *temporal predictability* (e.g., Worst Case Execution Time (WCET) analysis, scheduling) and temporal diagnosis shall at least be considered.

- *Spatial independence* is required to ensure that "data used by one element shall not be changed by another element" so "that elements will not adversely interfere with each other's execution behaviour such that a dangerous failure would occur" [13]. For example, exclusive access to resources is a common technique used to achieve this purpose.

The probability of dangerous failure per hour (PFH) depends directly on both the DC and the reliability (λ) of the device. PFH is lower with higher diagnostic coverage and lower λ (higher reliability). For a given safety function SIL, the standard [13] defines a system level average probability of failure range and the device is allocated a portion of this range (e.g., high demand SIL4: 10 FIT $> PFH >=$ 1 FIT). Equation 3 shows the simplest procedure to calculate PFH, for a safety system that puts the system in safe state on detection of any failure [13]. As a general rule, for complex components such as COTS multi-core devices, it is assumed that the probability ratios of dangerous and safe failures are 50% each ($\lambda_s = \lambda_d = 0.5 \times \lambda$).

$$\lambda = \sum \lambda_s + \sum \lambda_d \tag{1}$$

$$\mathrm{DC} = \frac{\sum \lambda_{dd}}{\sum (\lambda_{dd} + \lambda_{du})} = \frac{\sum \lambda_{dd}}{\sum \lambda_d} \tag{2}$$

$$\mathrm{PFH} = \sum \lambda_{du} = \sum ((1 - DC) \times \sum \lambda_d) \approx \sum ((1 - DC) \times 0.5 \times \lambda) \tag{3}$$

2.3 Reliability degradation threats

The latest manufacturing processes are already producing 7 nm devices thanks to the advances in CMOS downscaling technology, manufacturing processes and electronic design automation tools [5, 18]. Nevertheless, this CMOS technology downscaling leads to several threats that contribute to reliability degradation such as [32, 33, 34, 5, 35, 36]:

- *Process, Voltage and Temperature (PVT) variability*: As the technology continues to shrink, manufacturing process variability introduces higher variability of transistor / gates properties with respect to design time properties [32, 37, 38]. Runtime variability due to PVT and aging, can also lead to transistor / gates variability but at different locations and operational lifetime points of the device [32, 37]. This variability can lead to timing failure of circuits if design time margins are exceeded and therefore reduce the device reliability.

- *Transient faults and soft errors*: Soft-errors are caused by environmental conditions (e.g., α-particles, cosmic rays, ionizing radiation, EMI, cross-talk) leading to a transient perturbation that can be manifested as memory bit-flip in memory cells or combinatorial logic result error [32, 33, 39]. If the perturbation affects a single cell the event is called Single Event Upset (SEU), and if it affects more than one the event is called Multiple Bit Upset (MBU).

- *Permanent faults*: Permanent faults are those whose presence is persistent and permanently affect the correct functional behavior of the system [1, 33]. As the technology continues to shrink and device complexity continues to increase, the probability of physical defects in the hardware (e.g., short-circuit) during lifetime of a system continues to increase, from rare improbable events to probable events, and therefore error tolerance and diagnosis becomes also important to ensure the reliable and safe operation of the device [33].

- *Aging (degradation)*: The temporal degradation of the device can lead to timing degradation and permanent errors. As explained in [36] several factors contribute to this degradation, where *negative bias temperature instability* and temperature are prominent factors.

2.4 Semiconductor industry trend

As described by Corradi [18], the semiconductor industry is facing a technology and economic challenge, where fewer chip makers can afford the investments required for 16 nm and below technology. At industry level, the consequence is a reduction in the number of leading semiconductor companies and a general trend towards the mass-production of heterogeneous multi-core SoCs optimized for maximum average performance that target multiple domains with a single SoC solution [18]. Except for the automotive domain, safety-related semiconductor market niche is too small for semiconductor companies to provide economically competitive safety specific solutions compliant with previously defined certification standards. The automotive domain safety microprocessor solutions market is approximately $5 billion dollars [40, 19], approximately one percent of the global market estimated at $412 billion in 2017 [41].

Therefore, it is expected that multi-core devices used in the development of safety-related systems will have a higher dependency on fewer manufacturers with devices that tend to be either *safety device*, *generic device* or *hybrid device*. Table 1 provides a selection of mass-produced COTS *generic*, *safety* and *hybrid devices* provided by major embedded domain silicon manufacturers.

- *Generic device*: General purpose heterogeneous multi-core device optimized for cross-domain maximum average performance. As shown in Table 1, an average COTS *generic device* is designed for maximum average performance and organized in a shared resources architecture based on components such as [15]: cores, interconnect bus(es) and memory controllers, private and shared caches, Uniform Memory Architecture (UMA) / Non-Uniform Memory Architecture (NUMA) and addressable peripherals [20, 42, 43, 12, 44, 17, 18].

- *Safety device*: Multi-core safety device generally designed for the automotive domain that might also support industrial safety standards such as IEC 61508 [19, 12]. As shown in Table 1, an average COTS *safety device* is designed to comply with safety standard(s) and provide maximum average performance in a shared resources architecture based on components such as [15]: cores, shared buses, private caches, UMA / NUMA and addressable peripherals.

- *Hybrid device*: Multi-core device that combines previous options, e.g., *generic device* with a certifiable 'safety island' (e.g., Zynq UltraScale+), *generic device* with integrated FPGA that enables the integration of custom safety designs (e.g., Zynq with ARM and MicroBlaze [7]).

Device Type	**Core Family**	**Device Family**	**Manufacturer**
Generic devices	ARM	Zynq 7000	Xilinx
		Jacinto, Sitara, C6000	Texas Instruments
		R-Car V3H, RZ	Renesas
	Power Arch.	iMX8, QorIQ	NXP
	x86	Xeon, Atom C3000	Intel
Safety devices	ARM	Hercules (TMS570)	Texas Instruments
		R-Car H3	Renesas
	Power Arch.	MPC57xx	NXP
		SPC5	STMicroelectronics
	R850	RH850	Renesas
	Tricore	AURIX	Infineon
Hybrid devices	ARM	Zynq UltraScale+	Xilinx

Table 1: Summary of selected mass-produced COTS devices from major silicon manufacturers

3 Multi-core Device Architectures

3.1 Summary

As explained by Furano et al. [45], taking into consideration the limitations of available ASIC technology, the overall computational performance of devices can be increased by means of mono-core devices with higher frequencies, new architectures, Deep Submicron (DSM) and multi-core technologies. The scope of this survey are new architectures, DSM and multi-core technologies.

The developer of a safety-critical system based on multi-core device(s) needs to understand all relevant safety techniques provided by the given device and define required additional complementary measures to comply with applicable safety standards. However, due to the different nature of threats and challenges that must be addressed at different levels of abstraction (e.g., nanoscale manufacturing process variability, cache time predictability, cache coherence) there is a high research fragmentation and a great variability in terminology that makes this analysis difficult. This section contributes with a categorization of a wide and highly fragmented research area in the field of multi-core device architectures for dependable systems, based on three device abstraction levels (nanoscale, component and device) and a set of fundamental safety requirements that safety-critical system developers must at least address (see Section 2.2). Figure 1 shows the survey structure, mapping survey sections with device abstraction levels and fundamental safety requirements. Table 2 classifies selected research contributions described in this survey with respect to device abstraction levels and fundamental technical safety requirements.

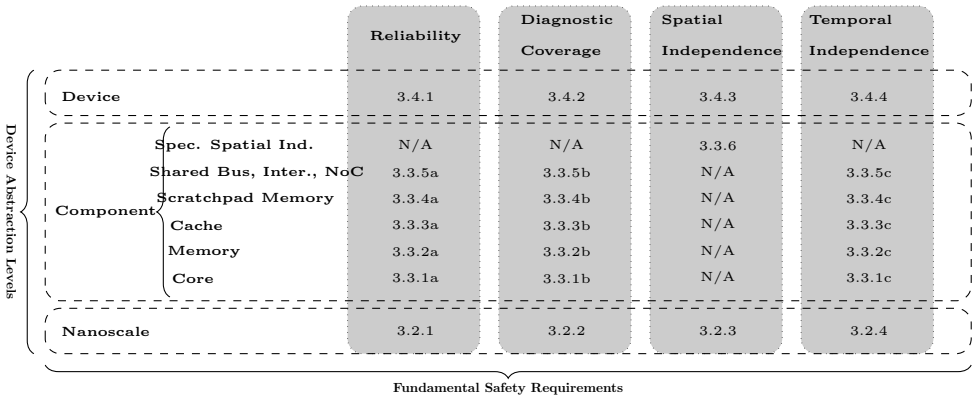

Figure 1: Structure of the survey

With respect to reliability and DC, several research surveys [5, 46, 39] and contributions [47, 36, 48, 49, 35] provide a comprehensive state-of-the-art of reliability threats and mitigation techniques, DC and fault-tolerance by means of error detection and correction techniques. Even within this specific topic, which focuses on reliability and DC techniques, there is a wide and partially fragmented research area that leads to terminology [5], taxonomy and classification variability. Nonetheless, selected representative techniques have been categorized with the previously described approach and this categorization can be used as a reference for the categorization of the additional techniques described in these contributions. Qualitative and quantitative comparison of techniques are also described in previously referenced contributions using similar (e.g., fault coverage, error coverage) but not always equivalent concepts with respect to IEC 61508 requirements for DC, thus not enabling a direct quantitative comparison of described techniques. Finally, an equivalent approach has been taken with respect to FPGA technology, where several research surveys [50, 51, 52] provide a comprehensive state-of-the-art of mitigation techniques, design standards and fault-tolerant methodologies for FPGAs.

Concerning temporal independence, Maiza et al. [15] provide a survey of timing verification techniques and time predictability challenges and solutions where selected component and architectural contributions have been categorized at component and device level. At nanoscale level, timing fluctuations are due to physical properties such as temperature and require different techniques. The development of time predictable components provides a solid foundation to achieve device level temporal independence, simplify WCET analysis and software time composability [53]. However, with respect to time predictability and WCET, most of the research contributions can be classified as either "how things should be done" (e.g., time predictable components) or "how things can be done" (e.g., how to achieve temporal independence with *generic devices*).

Regarding spatial independence, this is generally supported by specialized architectures (e.g., NUMA) and / or components (e.g., MMU, cache coherency) commonly available in COTS devices that can be used and configured by system designers. However, a major challenge is to ensure the correct configuration (e.g., multiple hardware data paths), reliability and DC (e.g., cache coherency diagnosis) of these components based on which spatial independence justification is based.

Finally, the current research fragmentation leads to a need to address a holistic cross-level and cross safety technical requirement approach as required to develop and certify safety-critical systems. This is observable in the development of domain specific case-studies described in Section 4.3, where fundamental safety requirements are addressed at different

Level	Requirement	Technical Contribution	Section
Device	Temporal Independence	- Software [59, 60, 7, 6, 4, 61, 62], WCET [63, 64, 65, 66, 67, 15, 68, 69, 58, 70, 71, 72, 42, 73, 74], scheduling [4, 59, 75, 74, 76] - Device architecture [77, 7, 78, 53, 79, 80, 72, 81, 82, 83, 84, 85, 62]	3.4.4
	Spatial Independence	- Software [59, 86, 7, 60, 54, 87, 88] - Device architecture: memory [74, 4], shared bus [7], cache [84, 88, 7], security [89, 90] - SRAM FPGA [91, 92, 93]	3.4.3
	Diagnostic Coverage (DC)	- Software [7, 4, 6, 94, 95, 89, 88, 7, 96] - Device architecture [7, 4, 6, 94, 89, 88, 49]	3.4.2
	Reliability	- Device architecture [97, 98]	3.4.1
Component	Temporal Independence	- Device components: core [85, 99, 100, 53, 101, 102, 103], memory [104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 47, 53, 115], cache [65, 67, 116, 117, 64, 118], NoC [119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130], SPM [131]	3.3.6 3.3.5 3.3.4 3.3.3
	Spatial Independence	See device level spatial independence	3.3.2 3.3.1
	Diagnostic Coverage (DC)	- Software [35, 46, 4, 6] - Device components: core [35, 39, 132, 48, 133, 37, 49], memory [46, 47], cache [46, 47], SPM [46, 47], shared bus [3, 7, 134], NoC [120]	
	Reliability	- Software: redundancy (e.g., spatial, temporal) - Device components: core [135, 136, 35, 49, 39]	
Nanoscale	Temporal Independence	- Software, device components and hardware: thermal and power management [137]	3.2.4
	Spatial Independence	- See component and device level reliability and DC - Mass-produced COTS device design process [13]	3.2.3
	Diagnostic Coverage (DC)	- Software [46, 4, 6, 35] - Device hardware [46, 35, 48, 36]	3.2.2
	Reliability	- Software [46, 4, 6, 56, 36, 137, 138] - Device hardware: [47, 38, 48, 36, 46, 35, 97, 98, 49] - SRAM FPGA [50, 51, 52]	3.2.1

Table 2: Selected research contributions classification by device abstraction level and fundamental safety technical requirement

abstraction levels using different application specific and ad hoc combinations of research contributions (e.g., wind turbine [7, 54, 6, 4, 55]). For example, a cross-level approach is required to achieve the DC required by safety standards because, as the complexity continues to increase for device architectures, components and interconnections, meeting the required DC at software application level becomes a challenge if sufficient hardware diagnostic support is not provided [7, 36, 6]. But, the addition of these diagnosis techniques in the hardware, can also have an impact on silicon die and lead to cost increase, power overheads and reduced performance. So, the combination of hardware and software techniques combined at different abstraction levels are usually required to detect complex hardware faults such as MBU and transient faults, as supported hardware techniques alone might not be sufficient [34, 39, 56, 48, 36]. However, cross-requirements research contributions that aim at reconciling reliability, DC and time predictability are still scarce (e.g., [57, 58]).

3.2 Nanoscale Level

At nanoscale level, selected key research contributions focus on the definition of techniques for reliability, diagnostic coverage, circuit spatial independence and circuit time predictability.

3.2.1 Reliability

Reliability laboratory testing data such as [139, 140] provide FIT rates and wear-out period estimations and measurements. Several surveys [36, 48] provide additional FIT rate data sources and analysis. Intuitively, as CMOS technology downscales

and supply voltage decreases, raw reliability should decrease (FIT increase) due to higher probability of random soft errors and transient faults because less energy is required for a cell upset event. In addition to this, reliability should also decrease due to increasing PVT variability, permanent faults and aging factors.

However, in multiple cases this trend is mitigated and even reverted as the technology downscales [36, 48]. For example, Xilinx provides reliability laboratory measurements where FIT rates for technologies between 350 nm and 16 nm vary between 2 and 12 FIT, with better values for 90 nm than for 350 nm [140]. This is due to several reasons such as to physical properties (e.g., sensitive deployment area decrease leads to memory soft error rate decrease [36, 48]), fabrication process improvements that mitigate PVT variability (e.g., lithographic process) and fault-tolerance techniques deployed in the device such as:

- *Design time fault-tolerance* that provides hardware level mitigation support without the run-time execution of additional external circuitry. For example, hardening (e.g., radiation hardening, resilient transistors against faults), design margin-based mitigation techniques (e.g., operating at higher supply voltages, gate sizing, body biasing, circuit guard banding) and diversity [47, 38, 48, 36, 49].
- *Run-time fault-tolerance* that requires the execution of additional circuitry. For example, redundancy (space, time, information), pipeline protection with shadow latches, tunable replica circuits [46, 35, 48, 36, 49]

For aging effect mitigation, thermal management (e.g., Dynamic Voltage and Frequency Scaling (DVFS), specific techniques to reduce temperature spatial and temporal gradients) and previously described circuit level mitigation techniques (e.g., guard banding) are common methods [36].

With respect to wear-out period, as explained in [138] and reliability laboratory testing data [139], device FIT rates fall sharply at the end of the chip lifetime. This wear-out period is reduced as the technology downscales [47] and imposes a relevant restriction in systems life-time, e.g., the typical wear-out period for 180 nm technology is higher than 100 years and for 65 nm it is expected to be lower than 5 years [138, 139].

Software Techniques As described in [35], previously described fault-tolerance techniques have an impact on silicon die and can lead to cost increase, power overheads and reduced performance. Therefore, devices targeting industrial markets with high reliability requirements are prone to include such features as oppose to high volume markets that have lower reliability requirements [38]. Because of this, application specific or generic software fault-tolerance techniques can be used in order to improve hardware reliability, e.g., multithreading for soft error fault-tolerance [56], *algorithmic based fault tolerance* [46], software redundant execution (e.g., software application, replicated instructions and check inserted by the compiler) [46], combination of software and hardware techniques with WCET guarantees [138], software level thermal co-management [36, 137], application specific techniques (e.g., [4, 6]).

Hardware Fault Tolerance of One (HFT = 1) IEC 61508-2 (Annex E) [13] defines common minimum requirements for the development of on-chip redundancy with hardware fault-tolerance greater than zero (HFT=1) using integrated circuits with one common substrate [13]. This solution enables the development of devices with on-chip redundancy (e.g., 1oo2) such as HICore1 [97, 98] and Zynq-7000 [141]. For that purpose, at least the following safety technical considerations shall be fulfilled at nanoscale and common substrate level [13]: separate physical blocks on substratum, common cause failure avoidance and mitigation (e.g., temperature, power supply), minimum distance between boundaries of physical blocks (e.g., $50\mu m$ [97]), short circuit and cross talk mitigation [13].

FPGAs SRAM FPGAs use a configuration memory that defines the operations of the electronic circuit implemented by the FPGA. Different hardening techniques can be used to tolerate and mitigate soft and transient errors that could lead to a modification of the intended circuit implementation, e.g., redundancy, scrubbing, partial dynamic reconfiguration, combinations of the previous techniques [50, 51, 52]. On the other hand, antifuse based FPGAs provide higher reliability than SRAM FPGAs, but their higher cost and lack of reprogrammability limits their application outside the aerospace domain.

3.2.2 Diagnostic Coverage (DC)

The run-time diagnosis of soft errors and aging errors is a challenge due to the random nature of the error, location and time distribution [32, 142, 7]. Multiple industrial and research grade techniques can be used to provide required circuit diagnostic coverage, e.g., monitoring circuits, self-tests and diagnostics, transition detector with time-borrowing, code-based techniques [46, 35, 48, 36]. Example techniques are described at core and memory component level in Sections 3.3.1 and 3.3.2.

3.2.3 Spatial Independence

Non intended electromagnetic interference between nanoscale level elements such as signals, clocks and power-supplies due to common cause failures such as cross talk can lead to component and device level errors. These faults are mitigated by state-of-the-art mass-produced COTS device design processes [13], reliability and DC techniques described at component and device level (see Section 3.4.3).

3.2.4 Temporal Independence

Device voltage and temperature fluctuations can lead to circuit timing fluctuations. For example, in specific designs 10% supply voltage variation can lead to up to 20% circuit delay and 10 °C temperature increase can lead to approximately 5% interconnect delay [35]. While temporal diagnosis techniques can detect and react / recover to circuit timing errors, this is a potential temporal interference source among integrated functions. For example, the computation and resource usage of a non-safety function could potentially generate a temperature increase that leads to minor timing fluctuations in the computation of a safety function. In addition to this, as described in [137], device built-in mechanisms for thermal and power management (e.g., DVFS) that could lead to temporal interference among integrated mixed-criticality functions should be analyzed during design time and safely managed during run-time.

3.3 Component Level

At component level, key research contributions focus on the definition of techniques for reliability, diagnostic coverage and time predictability for all common device components such as core, memory, cache, scratchpad memory, shared bus, interconnection networks (e.g., Network-on-a-Chip (NoC)) and specialized components for spatial isolation (e.g., Memory Management Unit (MMU)).

3.3.1 Core

A core is a program processing unit that reads and executes program instructions (software application) and reads / writes data in associated memories.

Reliability The survey [39] and overviews [35, 49, 36] describe and provide quantitative comparisons of processor core fault-tolerant architectures based on techniques such *hardening* and *redundancy*. For the given examples and compared techniques [39, 35, 36], hardware replication can generally provide the highest reliability increase and generally it has a higher hardware overhead (e.g., >= 300% for triplication) than hardening techniques (e.g., < 15% for LEON3FT).

- *Hardening*: LEON3FT and LEON4FT are 32-bit SPARC-V8 RISC architecture soft-cores designed specifically for space domain fault-tolerance, based on an extensive usage of error detection and correction codes such as ECC for memory blocks [135].
- *Redundancy* (triplication): ARM Cortex-R5 based triple core lock-step architecture provides a triple modular on-chip redundancy solution with an error propagation unit that includes majority voter, error detection logic and synchronization logic [136]. With respect to software, software triplication schemes with voting can also be deployed among on-chip redundant cores (e.g., on-chip redundancy with HFT = 1 [4]) to tolerate soft errors but might not be valid to tolerate permanent errors.

Diagnostic Coverage (DC) The survey [39] and overviews [35, 49] describe and provide quantitative comparisons of several online error detection and recovery techniques based on techniques such as 'redundancy', 'dynamic verification', 'anomaly detection' and 'temporal diagnosis' for which selected contributions are summarized:

- *Lockstep redundant execution* provides a safety standards compliant (e.g., ISO 26262, IEC 61508) medium to high DC with a high hardware overhead cost (e.g., >= 200%) and minimum performance overhead [35, 49]. This is a cost-competitive and common technique used in COTS *safety devices* listed in Table 1 (e.g., AURIX TC3xx). However, dual-core devices that operate in locked-step, in order to increase the core DC are considered single-core devices.
- *Dynamic verification* uses dedicated run-time hardware checkers, which are not redundant execution units, to validate specific execution invariants that are assumed to be true in the absence of errors [35]. Analyzed hardware based techniques have low hardware overhead (e.g., $6\% - 17\%$), low detection latency and low performance overhead [35, 39]. Argus [132, 35] is an example hardware technique that can potentially achieve medium DC with low hardware area overhead (< 17%) [35]. It performs core diagnosis by means of run-time checking of four invariants : the control flow and data-flow are executed as specified in the program binary, correct computations are performed, and memory is not corrupted. It does not perform low level checking of core components nor provide redundancy of cores for diagnostic purposed. Generic software techniques such as control flow checking [13, 49] can be used to diagnose that the expected control flow is executed (the invariant). Specialized software-based redundant solutions can provide medium potential DC with no hardware overhead cost, low latency but high performance overhead [35]. For example, *software implemented fault tolerance* is a soft error diagnosis solution that integrates control flow checking with compiler-based transformation of the software that duplicates program instructions and inserts checkpoints to diagnose incorrect computations. Arithmetic codes [49] are also specialized software redundant solutions that add data redundancy in order to generate codewords that if correctly processed by the associated arithmetic operations will also generate a codeword (the invariant).

- *Anomaly detection* techniques use dedicated or modified circuits to detect errors. For example, periodic Built-In Self Test (BIST) can potentially achieve low to medium DC with low hardware overhead (5%) [35]. Software based techniques do not require hardware overhead but require a relevant software overhead to perform periodic tests. For example, a periodic *software-based self-test* may require $5\%-25\%$ of system execution time [35]. Other example software techniques are assertion and sanity-based fault-tolerance [46] and application specific techniques (e.g., [4, 6]).

- *Temporal diagnosis* techniques use dedicated or modified circuit to detect timing errors. For example, Razor provides circuit-level error detection and correction of timing errors with low hardware overhead (e.g., $< 3\%$) [133, 37]. It targets the development of power reduction processors where the error rate during operation is monitored to tune the supply voltage.

Temporal Independence Table 1 provides a summary of selected COTS core families for *generic*, *safety* and *hybrid devices*. In this summary, it is possible to conclude that a limited set of core families are predominately used by major device manufacturers (e.g., ARM, Power Architecture, x86). These generic purpose complex cores provide high average-case performance but time predictability and WCET estimation becomes a technical challenge as described in Section 3.4.4.

On the other hand, as described by Schoeberl et al. [85], a relevant research activity targets the development of time predictable cores that potentially simplifies the WCET estimation and provides lower bounds. The publication [85] provides an overview of time predictable cores, from which most relevant time predictable cores with respect to the scope of the survey are summarized and extended with additional research contributions of interest:

- Patmos [85] is an open-source RISC processor core designed for time predictability and low WCET bounds. The design includes a dual-issue pipeline, specially designed caches for WCET analysis, support for scratchpad memories and NoC interface.

- FlexPRET [99, 100] is a multithreaded RISC-V [143] processor designed for the integration of hard and soft real-time threads, which includes a hardware thread scheduler to schedule hard real-time threads and soft real-time threads. It is an extension of the Precision Timed (PRET) processor [101, 102, 85] that implements ARM ISA with a RISC pipeline, chip-level multithreading, Time-Division Multiplexing (TDM) to access shared main memory and scratch memory instead of caches.

- Hard real-time SMT core [53] implements Tricore instruction set [21] with two in-order super-scalar processor pipelines (integer and address) with multithreading support. This core is designed to ensure task level bounded maximum delay due to inter-task interference, between hard real-time and non-hard real-time tasks.

- SPEAR [103] is a constant-time instruction set processor with minimum interrupt jitter and delay support. It is based on a custom instruction set with fixed and constant execution time.

- PTARM [101] is a soft-core that implements a subset of the ARMv4 ISA with a thread-interleaved pipeline and provides a time predictable DRAM controller.

3.3.2 Memory

The execution of software applications in cores requires program and data memory storage. As explained in [144, 12], in multi-core devices two basic memory architectures are used: Non-Uniform Memory Architecture (NUMA) and Uniform Memory Architecture (UMA). In NUMA each core has a dedicated physical memory with exclusive access privileges and UMA is a shared memory architecture where the physical memory is uniformly shared among cores and additional peripherals such as Direct Memory Accesses (DMAs).

Reliability As explained in [36, 47], memory capacity continuously increases with an associated increase in the amount of occupied silicon area and overall device reliability weight. While memory soft error rate decreases due to downscaling because the sensitive deployment area decreases, the integrated memory sizes continue to increase at a greater pace [36, 48]. In order to improve memory reliability several new technologies have emerged such as phase change memories [47].

Diagnostic Coverage (DC) As described by [46, 47] and supported by COTS devices described in Table 1, error detection in memory is generally performed by information redundancy techniques such as parity bit (low DC) and Error-Correcting Codes (ECCs) that can provide medium DC.

Temporal Independence Conventional *memory controllers* are not designed to support temporal independence for mixed-criticality software applications and this leads to potential temporal interference between applications that share the same *memory controller*. A conservative solution to bound the temporal interference and WCET, is to consider the worst-case latency for all possible memory transactions. However, this solution focuses on the WCET scenario, ignores application and device specific solutions to improve overall efficiency (e.g., software applications scheduling, bandwidth utilization) and the

unpredictability of applications dense memory access patterns. Furthermore, it is more difficult to provide a tight WCET for multi-core devices compared to single-core devices, due to inter-application interference accessing shared resources. As a consequence, the execution time of a given software application depends on other simultaneously executing applications [145]. In order to support time predictability, specialized memory controllers and management strategies can be used, for example:

- In order to enhance efficiency, a dynamic command scheduling based back-end real-time memory controller architecture was proposed [109]. A conservative open-page policy is proposed by [115] that improves average-case performance (e.g., bandwidth, latency) of a firm real-time controller while supporting real-time guarantees.
- [107] offers an interference-free memory for critical applications with a memory controller that supports the integration of applications of different time criticality. In this approach, memory is treated as a set of independent virtual devices that are mapped to mixed critical workloads by a partitioning strategy.
- [108] proposes a memory controller that supports dual-criticality by means of virtual division of memory banks. In this approach, each virtual memory bank type is managed with a specific request scheduler policy.
- MemGuard [104, 106, 146] provides an algorithm to perform memory bandwidth management at core level by using hardware performance counters to monitor the number of last-level cache misses, or accesses via the memory interconnect. [147] introduces a bandwidth regulation module that adapts and extends the MemGuard Linux kernel module algorithm [105] (known as MemGuardXt) for hardware implementation and supports specific operating modes and options, with guarantees for rate-constrained flows and bandwidth rates.

Transactional Memory (TM) controller is a concurrency control mechanism for shared resources. It has been introduced to simplify the concurrency management in multi-core systems, by supporting the atomic execution of sets of load and store instructions. They suit safety-critical systems as they support time predictability and fault isolation. The *temporal and spatial partitioning* approach is commonly used to establish the required determinism in the implementation of transactional memories. A transactional memory controller can be implemented as software (STM), hardware (HTM) or combination of both. The following selected contributions focus on the WCET pre-estimation for the schedulability of a given software application(s) task set:

- [110] introduces an STM algorithm that prevents starvation by means of update transactions conflicts management. The selection of the conflict detection policy (lazy or eager type), has a significant effect on the temporal schedulability because it determines when conflicting transactions are aborted. [111] presents a real-time scheduling perspective analysis of STM conflict detection policies. [112] presents an STM concurrency control for multi-core devices with real-time software applications. It also presents transactional conflicts contention manager.
- [53] describes a dynamically partitioned cache interface solution that handles memory transactions with bounded maximum delay guarantees for high criticality software tasks (e.g., hard real-time tasks). This solution, executes on top of the memory controller. In [114] an analysis of conflicts and aborts for different criticality transactions is provided (e.g., hard real-time, best-effort). Moreover, [113] describes a Software Transactional Memory (STM) contention manager that supports priority-based transactions.
- [148] describes a priority-based HTM controller designed specifically for mixed-criticality systems, with support for timing analysis, selection of conflict resolution algorithms and techniques for fault propagation avoidance.

Previously described approaches address the resolution of memory transaction conflicts. However, they do not avoid or address temporal interference in other components such as the interconnect (see Section 3.3.5).

3.3.3 Cache

Cache components [149, 150, 151, 152] reduce memory access time of computing cores, bridging the frequency gap between cores that operate at higher frequencies than memory components. A multi-core device might integrate private caches assigned exclusively to given cores (e.g., L1 cache) and public caches shared by several components such as cores (e.g., L2 cache).

As explained in Section 2.4, an average COTS *generic device* integrates a hierarchy of private and shared caches: one or several private caches per core connected to interconnect(s) and shared caches between the interconnect(s) and shared memories. An average COTS *safety device* integrates a private cache per core connected with the shared bus where memory and addressable peripherals are connected. Few exceptions (e.g., NGMP space multi-core device [153]) have shared caches that can be configured to operate partitioned across cores so that each partition effectively becomes a private cache. On the other hand, research devices that target the development of time predictable devices tend to use different approaches (e.g., T-CREST [77, 85]) and NoC based solutions with private caches (see Section 3.3.5).

Reliability Most of the area in computing devices is devoted to SRAM storage in the form of cache memories. Hence, their reliability is of prominent importance. Increased integration has led to increased susceptibility to transient faults, which are mostly addressed with parity and ECC [46, 47]. The particular solution to use depends on whether data is stored redundantly, so that parity suffices for error correction, or data is unique and ECC is needed for recovery. The existing options are discussed by Benedicte et al. [154].

Diagnostic Coverage (DC) DC in caches often builds on parity and ECC [46, 47], already deployed for reliability purposes as discussed before, since they allow testing contents correctness on each access at speed. On the other hand, cache coherency diagnosis is described at device level (see Section 3.4.2)

Temporal Independence Although caches improve average performance and WCET, it is hard to prove in general whether cache accesses will be guaranteed to hit, either with abstract interpretation or in a test campaign [65, 67]. Hence, time guarantees are hard to obtain in general, and so caches defeat either WCET estimates tightness or trustworthiness. Moreover, speculation and priority inversion effects in computing devices may make cache hits – whose latency is lower than that of misses – lead to longer execution times than cache misses, thus further challenging time predictability [116].

A thorough survey on cache designs and time predictability can be found in [117]. Details on the pros and cons of cache memories for probabilistic timing analysis is further developed by Cazorla et al. [64]. In this context, Valsan et al. [118] analyze the impact on time predictability of resources surrounding cache memories, such as queues and buffers, and noticing that, if not managed properly, cache (space) partitioning may not suffice to achieve time predictability.

3.3.4 Scratchpad Memory (SPM)

SPMs are software-managed memories, thus being interesting alternatives to traditional, hardware-managed, caches [131, 155]. SPMs are usually small and high-speed SRAM. SPMs reduce design complexity, and thus improve power and performance, w.r.t. cache memories, and improve time predictability by granting users and compilers with explicit control of their contents [131]. SPMs disadvantages relate to the burden of explicitly controlling their contents, which increases software development costs and may jeopardize software portability, and thus the ability to reuse legacy software. Multi-core safety devices like the AURIX TriCore-based architecture include private both program and data SPMs for each core in combination with instruction and data caches [21]. Hybrid devices such as the Zynq UltraScale+ also include an SPM, named on-chip memory, shared across the different processor cores in the platform [22].

Reliability As in the case of caches, SPMs are equally vulnerable to transient faults, and hence, solutions such as coding (in the form of parity or ECC [46, 47]) are highly popular for SPMs.

Diagnostic Coverage (DC) As in the case of caches, parity and ECC already provide means to reach required low to medium DC [46, 47]. Moreover, differently to caches, coherence is not a concern for SPMs since their contents are explicitly managed by software [131, 155].

Temporal Independence Time predictability is a key factor that eases WCET estimation and therefore temporal independence, and a key feature of SPMs is that they provide predictable access times. In multi-core architectures with SPMs like the AURIX and Zynq UltraScale+ [21, 22], the SPMs allow achieving higher levels of temporal independence than with cache based solutions, as they remove the impact of cache hit/misses and the cache coherence problem.

3.3.5 Shared Bus and Interconnection Network

The integration of multiple cores and shared peripherals in a single device requires the integration of inter-core communication mechanisms and shared access to peripherals with ideally low latency and high bandwidth. Different architectural patterns with specific on-chip communication solutions are applicable depending on the device type and number of cores: shared bus, interconnect (or switch) and NoC. Each architectural pattern has different reliability, DC and time predictability characteristics.

Reliability *Interconnect and switch* components are proprietary solutions with none or limited detailed documentation provided by silicon manufacturers [24]. Thus, component reliability measurements are provided by the manufacturer and limited information is known about internal fault-tolerance mechanisms. However, error detection and correction for data transmitted is possible by building on end-to-end ECC protection, i.e., codes are sent along with the data. However, such a solution does not provide by itself support to detect deadlocks, livelocks, lost message detection and the like. With respect to NoCs, the survey [156] provides a summary and quantitative comparisons (e.g., bandwidth) of relevant contributions for reliability and fault-tolerance in real-time NoCs.

Diagnostic Coverage (DC) A shared-bus provides a simple solution already considered by current safety standards, with known failure modes and associated diagnosis techniques [13]. Interconnects, on contrast, are complex solutions with none or limited detailed documentation [24] and consideration by safety standards. Due to this, direct diagnosis becomes a challenge and several related research contributions consider it a 'black channel' on top of which a Safe Communication Layer (SCL) can be deployed to support the safe communication among software tasks running in different cores [3, 7, 134]. A SCL can be used to provide a high DC of all possible communication errors that can occur in a 'black channel', the interconnect or switch. With respect to NoCs, the survey [156] provides a summary of relevant contributions for fault detection in real-time NoCs.

Time Predictability Generally, a *shared-bus* based on a round-robin scheduling policy has a a moderated impact on WCET variability and analysis complexity [157, 158]. But due to latency and bandwidth limitations, a shared-bus is generally used to interconnect a reduced set of cores. As shown in Table 1, a generic COTS *safety* device integrates 2 - 3 cores with a shared bus.

Interconnects integrated in COTS devices (e.g., P4080 CoreNet [3]), on contrast, have a considerable potential impact on WCET variability and not possible to limit due to the complexity and limited information available for analysis [24, 159]. As shown in Table 1, a *generic* COTS device integrates 2 - 8 cores with interconnect and switch solutions. The survey [15] describes relevant contributions for temporal predictability and WCET based on interconnect components.

On the other hand, as explained in the survey [120], a relevant research activity targets the development of real-time and time predictable *NoC* solutions for which a generic survey and quantitative comparison is provided by [120, 156]. With respect to NoC communication traffic temporal predictability, two major technical approaches can be defined: full and virtual traffic separation.

Full traffic separation is the most conservative approach to establish time predictability for safety-critical subsystems by completely separating the safety-critical and low-critical traffics. In this approach, the separation of traffics is done in the spatial domain. A subset of resources is reserved for the safety-critical subsystems and no requests from low-critical subsystems will be handled by this subset. However, this approach introduces a low-efficient usage of the network resources, as some resources are blocked for safety-critical subsystems. For example:

- *Fully Disjoint Routes*: Hermes [123] is an example, which introduces a distributed fault-tolerant routing algorithm and utilizes load-balancing routing. This routing algorithm provides pre-configured alternative path selection for fault-tolerance and bypasses faulty network path / region / areas [123]. However, the performance degrades with the number of faulty links.

- *Circuit Switching*: Programmable NoC (PNoC) [124] is a lightweight flexible circuit-switched NoC, which supports a high communication data-path width, dynamic insertion / removal of nodes and guaranteed throughput with low communication jitter. However, although the authors tried to minimize the circuit establishment latency using simple communication protocols, this delay is not deterministic because it depends on the availability of the resource.

- *Multiple Overlay Networks*: TilePro64 [125] is a complete SoC that offers distributed shared resources (e.g., memories, network controllers) using a packet-switched, wormhole-routed, point-to-point network. The on-chip interconnect network [130] establishes tile-to-memory, tile-to-tile and tile-to-IO communication by transmitting packets across the network.

In *virtual traffic separation*, temporal partitioning is used to virtually establish the separation between the safety-critical messages and low-critical traffic. The aim of this separation is to eliminate interference between applications and to achieve composable services. This approach can be implemented by defining scheduled injection times (e.g., Time-Triggered solutions) or subdividing the bandwidth between the mentioned traffic types using TDM approach (e.g., Ætherial). In addition, rate controlling (e.g., Nostrum) and prioritized virtual channels (e.g., QNoC) can be used to enhance the interference between two traffic types. The first two approaches (i.e., scheduled injection time and TDM) rely strongly on a notion of global synchronicity. For example:

- *Time-Triggered Network-on-a-Chip (TTNoC)*: In time-triggered solutions, the transmission of messages is performed according to a predefined communication schedule in a periodic manner with a constant period and at a defined instant within the period (the phase) [122]. TTNoC [126, 122] introduces an on-chip time-triggered interconnect that supports the required time predictability and fault-tolerance for mixed-criticality systems. However it lacks support for the integration of legacy hardware and transmission of event-triggered messages.

- *Fixed TDM Traffic Slotting*: The Ætherial [127] NoC supports two distinct traffic classes of communication, Best-Effort Services (BES) and Guaranteed Services (GS). The GS offers uncorrupted, loss less, bounded latency, guaranteed throughput and ordered data delivery through resource (wires and buffers) reservation. This reservation is however according to the worst-case. The BES does not reserve any resources and uses unallocated and unused capacity and operates based packet-switching flow control. In this context, GS establishes a time predictable communication infrastructure for safety-critical subsystems, while BES increases the resource efficiency and suits low-critical subsystems.

- *Priority-Arbitrated Virtual Channels*: The Quality-of-Service NoC (QNoC) [129] offers multiple levels of services (Quality of Service (QoS)) and uses simple architecture by a credit-based back-pressure flow-control. Packets associated to different classes of service (GS, BES) are delivered in an interleaved manner with associated QoS definition priorities such as throughput, relative priority and end-to-end delays. Although packets of high priority incur a low delay, low-priority flows suffer interference from higher priority flows [121].

- *Rate Controlling*: Rate controlling, or bandwidth management, is the process of measuring and regulating the communication (traffic, packets) on a network link to avoid overloading the link, which leads to network congestion or poor performance. This technique obviously cannot eliminate the interference but minimizes it. Nostrum [128] defines a packet-switched 2D mesh topology, which offers GS as well as BES using deflecting routing algorithm. It supports design time bandwidth guarantees on a path by means of prioritized looped containers.

3.3.6 Specialized Components for Spatial Independence

In order to achieve multi-core device level spatial independence in compliance with IEC 61508, several specialized components for spatial independence can be used, e.g., MMU. These components are described in Section 3.4.3.

3.4 Device Level

This section describes selected key research contributions towards device level reliability, diagnostic coverage, spatial independence and temporal independence.

3.4.1 Reliability

As explained in Section 2.2 and Equation 1, the device failure rate is composed by the addition of safe and dangerous failure rates of all components, or at least the components that take part in the execution of safety functions. Thus, device level reliability is defined by the device architecture and building components reliability (e.g., Reliability Block Diagram (RBD) with serial / parallel composition of components).

Specialized device architectures such as HFT=1 (see Section 3.2.1) support on-chip redundancy with hardware fault-tolerance of one (e.g., 1oo2) [97, 98, 141], with a potential reliability improvement with respect to standard multi-core devices (e.g., 1oo1 (single channel)).

3.4.2 Diagnostic Coverage (DC)

This section describes several diagnosis techniques specific to multi-core devices: temporal diagnosis, spatial diagnosis and safe start-up / shut-down.

- *Temporal diagnosis* is required to diagnose that safety related time constraints of interest are met and safely react / recover in case of error. For example, in *safety devices* it is required to detect unexpected errors due to systematic and random errors, and in *generic devices* for detecting rare event situations not considered in the design phase. For this purpose, several common application specific techniques can be used (e.g., time constraint monitoring, trapping unexpected interrupts, watchdog [7, 4, 6]) and run-time monitoring diagnosis (e.g., device performance counters [94], software framework [95]).

- *Spatial diagnosis* is required to ensure that data of a given function is not modified by another element or function. In a multi-core device, the parallel execution of software tasks that perform read and write operations in memory require a coherency management, to ensure that a write operation of a given data value is consistently updated for read(s) operation by tasks in other core(s) within a given time frame. Cache coherency management is generally performed by dedicated built-in components (e.g., Snoop Control Unit [88]) that ensure the coherency of distributed and hierarchical caches (e.g., L1 / L2). Device built-in cache and memory coherency management are subject to systematic and random faults, thus, diagnosis is required to ensure spatial independence. Cache and memory coherency diagnosis can be performed by built-in hardware diagnosis [49], generic software techniques (e.g., [88, 7]) or a combination of both. In addition to this, device built-in spatial protection components (e.g., MMU) and error detection and correction techniques (e.g., ECC) require periodic diagnosis to detect random or systematic faults in these components and techniques [4, 89, 96]. Finally, application specific software techniques based on information redundancy for application safety data areas might be used to increase the spatial diagnostic coverage (e.g., CRC, memory duplication, memory duplication with bit inversion [13]).

- *Safe start-up and shut-down* diagnosis is required to ensure that the device is started-up and shut-down in a safe, design time defined and repeatable manner [4, 89], using internal and / or external hardware and software,

3.4.3 Spatial Independence

Key contributions with respect to device level spatial independence can be classified in two major groups: achieving spatial independence with shared resources and specialized architectures and components that support spatial independence. Spatial diagnosis techniques and associated components are also required to detect errors, must be considered in reliability analysis and temporal independence analysis (see Sections 3.4.2 and 3.4.4).

Spatial Independence with Shared Resources Ensuring exclusive access to addressable resources is a common basic technique to ensure spatial independence avoiding the threats associated to shared resources. In UMA and NUMA memory architectures where addressable resources are shared resources, the usage of MMU and Memory Protection Unit (MPU) components is a common technique to support the exclusive access to shared resources such as addressable peripherals and memories [74, 4]. Both components are common in state-of-the-art multi-core devices (see Table 1). Moreover, in *generic devices* shared components might be accessible by means of different buses and networks, thus, possible combinations shall also be analyzed and restricted [7]. In an UMA memory architecture with shared memories and local cache per core, coherency threats need to be mitigated managing the device level coherency of shared data read and write transactions. When a shared data is locally updated, this change needs to be timely updated at device level by means of device specific cache coherency support [88, 7].

In addition to this, modern multi-core devices provide built-in security technology that in some cases can be used to support spatial independence. For example, TrustZone technology provides device level hardware isolation for trusted software, which can be used to define a trusted execution environment for the safety software with exclusive access to assigned resources [89, 90].

Specialized Architectures and Components for Spatial Independence In a NUMA memory architecture, where each core has a dedicated physical memory with exclusive access privileges, spatial independence of local memory can be potentially achieved by design [12]. This is a common approach used in NoC architectures such as specialized time independent architectures described in Section 3.4.4. This approach is also used in HFT of 1 solutions (e.g., [97, 98, 160]).

Device virtualization features enable the development of software level solutions that support spatial independence among software partitions and tasks. For example:

- A hypervisor is a layer of software that uses device virtualization support in order to provide independent execution environments to software partitions (e.g., XtratuM, Linux-KVM, PikeOS, Jailhouse [59]). Hypervisor virtualization in combination with multi-core devices can be used for the development of mixed-criticality systems combining safety and non-safety partitions, including safety partitions of different criticality. The development of hypervisor solutions for mixed-criticality applications is an active field of research [59, 54, 7, 60] in combination with the usage of such technology with multi-core devices for the development of mixed-criticality safety systems [161, 4, 6, 3, 2, 162]. Gu [60] provides a survey of hypervisor virtualization solutions for real-time and safety-critical systems. Domain specific safety certified COTS hypervisors are already commercially available (e.g., PikeOS, XtratuM).

- With respect to *real-time operating systems*, several research initiatives are analyzing container-based mechanisms, which are well-known in the security domain, to provide software isolation mechanisms similar to previously defined software partitions that can also be deployed in multi-core device architectures (e.g., [87]). For example, several initiatives aim to pave the way towards safety certification of Linux (e.g., SIL2LinuxMP [86, 87]).

In addition, and based on this, specialized software strategies can be defined to handle specific spatial independence challenges such as shared addressable registers used to manage input, output and communication peripherals. If more than one function requires the shared usage of such peripheral registers, previous strategies to support exclusive access to memory addressable resources might not be valid. For example, digital input / output and communication servers describe software solution patterns based on partitions that exclusively manage device registers and provide read / write services to other software partitions [7, 88, 84].

With respect to FPGAs, modern SRAM FPGAs provide different logical and spatial isolation mechanisms and supporting tools. FPGAs with multiple dies on a substrate can support the route and placement of redundant channels in different silicon die blocks [93]. In addition to this, isolation fences for spatial independence can be defined in order to provide logical and spatial isolation between logical blocks using specialized qualified tools such as Isolation Design Flow (IDF) [91, 89].

3.4.4 Temporal Independence

Key contributions with respect to device level temporal independence can also be classified in two major groups: achieving temporal independence with shared resources and development of specialized architectures that provide or support temporal independence. In both cases, WCET estimation is required for timing verification [15] and temporal diagnosis to detect errors (see Section 3.4.2).

Temporal Independence with Shared Resources As explained by Maiza et al. [15], Mitra et al. [72] and in Section 2.4, an average COTS device is based on a shared resources architecture. The concurrent access to shared resources and components that optimize average performance at the cost of time predictability, leads to potential temporal interference among executing software tasks that could jeopardize the required temporal independence [42, 43, 163, 116, 4, 17, 15, 72, 10] and WCET estimation with low bounds becomes a challenge [67, 15, 72].

Nonetheless, as explained in [4, 6, 7], it is feasible to define application and device specific time independent solutions, using COTS *generic devices* with shared resources and associated intrinsic time interference sources [42, 72]. In order to achieve temporal independence, at least a design systematic approach and diagnostic coverage to detect temporal constraint violations are required. If an unexpected dangerous violation occurs, the diagnosis error reaction should lead to safe-sate, which does not jeopardy safety but availability. The design systematic approach should at least consider analysis of device and components temporal characteristics such as WCET estimation, partitions scheduling, partition time slot assignment and interrupt sources management. The temporal diagnosis strategy should consider techniques such as monitoring of time constraints and trap unexpected interrupts. As summarized in Table 3, this approach has been extended in several domain and application specific safety concepts, e.g., railway [3], automotive [2].

But, this approach considers a simplistic scenario where a single fail-safe safety function is integrated. As explained by Paulitsch [74], when several fail-safe functions are integrated the requirement for function integrity and availability needs to be analyzed to ensure safe operation. A safety function might not be allowed to sacrifice the operation of another safety function due to the unforeseen consequences of the effect (e.g., cascading effects to other functions that could lead to dangerous pilot cognitive overload) [74]. Thus, these scenarios require also application and device specific solutions to achieve the required temporal independence.

Specialized Architectures for Temporal Independence Several research contributions define specialized architectures and devices that provide rules, patterns and building-blocks that can be used in combination with application software to achieve temporal independence:

- The Composable and Predictable Multi-Processor System on Chip (CoMPSoC) architecture [78] supports resources between applications on the same multi-core system-on-a-chip, while avoiding mutual interference. The temporal behavior of an application does not depend on other applications despite the sharing of on-chip resources. CoMPSoC employs the Ætherial NoC to ensure predictability for the communication. This NoC [82] uses time-division multiple-access and static resource allocation for the network interfaces, where time-slots remain empty in case no data needs to be transmitted by a network interface.

- The DREAMS architecture [7] provides structuring rules according to several integration levels. At the chip-level, a multi-core chip is decomposed into tiles that are interconnected by a NoC, where each tile is composed by several cores with associated caches, memories and I/O resources. The architecture defines generic platform services as a baseline for the development of applications including a integrated resource management and fault-tolerant global time base, based on which secure communication and execution of services are deployed. The architecture avoids interference of shared memory resources using MemGuard with a dynamic reclaim mechanism [147]. Each tile is allocated a certain bandwidth for each period and unused bandwidth is assigned to a global repository, which can be exploited by other cores.

- T-CREST is a time predictable multi-core architecture based on Patmos time predictable processor core, specially designed cache for WCET, scratchpad memory and time predictable TDM based NoC [77, 83, 81, 85].

- The Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability (parMERASA) architecture [79] provides an execution environment for hard real-time applications on a scalable multi-core processor. The parMERASA multi-core processor contains scalable timing analyzable NoCs. Processor cores are organized into clusters that are connected with a dedicated NoC [62]. It includes an extended on-demand coherent cache controller [84] that enables tight static WCET analysis [73]. This cache prevents unintended interference between cores by distinguishing between private and shared cache lines.

- The LEON-based Probabilistically Analyzable pRocessor Design (LEOPARD) architecture [80], implemented in a commercial LEON3-based multi-core for the space domain – although applicable to any domain – provides specific timing properties needed for probabilistic timing analysis. In particular, LEOPARD randomizes shared bus arbitration and cache placement and replacement policies. It also upper bounds the latency of some input dependent floating-point operations such as division and square root. This facilitates probabilistic timing analysis [64] and relieves developers from having to control design details such as memory placement and how tasks in different cores share hardware resources.

Worst Case Execution Time (WCET) The WCET estimation challenge can be viewed from different angles, each one with different paradigms to address the challenge:

- *Static vs measurement-based timing analysis.* Static timing analysis builds upon a timing model of the device where real-time applications should be executed, and performs abstract interpretation of the execution of the software on the device. This way, static timing analysis determines or, quite often, upper bounds the latency of the instructions, basic blocks and functions of the application under analysis on such a device. Static timing analysis reliability and tightness strongly depend on the accuracy and reliability of the timing model of the device, as well as on the amount of – detailed – information obtained from the application under analysis, mostly related to memory placement and timing of the events. Conversely, measurement-based timing analysis relies on actual execution time measurements of the application running on the actual device, thus avoiding any reliability and tightness issue due to models and detailed information for abstract interpretation. However, measurement-based timing analysis reliability strongly depends on the representativeness of the tests executed w.r.t. the WCET, as well as on the control exercised on the initial conditions of the device.

- *Deterministic vs probabilistic timing analysis.* Deterministic timing analysis aims at providing an absolute WCET bound that cannot be exceeded under any circumstance. However, the residual risk of exceeding that bound is unknown and not null by construction. Conversely, probabilistic timing analysis aims at delivering a probabilistic distribution intended to upper bound the real execution time distribution so that for any given exceedance threshold (e.g. 10^{-12} per run) a probabilistic WCET bound is obtained. In general, the latter, probabilistic timing analysis, offers a more convenient and flexible trade-off to obtain WCET estimates, but relies on some timing characteristics of the device and/or the application that may not hold unless appropriate hardware or software support is provided.

A plethora of works have analyzed and reviewed the different WCET estimation paradigms. The seminal work by Wilhelm et al. [65] compares static and measurement-based approaches, and hybrids thereof. An updated taxonomy of the state-of-the-art with focus on multi-core aspects for timing analysis can be found in [66]. A comparative perspective of all paradigms above with specific discussion for multi-core devices has been given by Abella et al. [67], whereas a deeper analysis for multi-core aspects can be found in [15]. Puschner [68], instead, considers the multi-core problem with particular focus on the management of mixed criticalities. Finally, tooling aspects for the different paradigms have also been specifically addressed [65, 71].

Measurement-based (deterministic) timing analysis has been object of significant attention with the development of multiple methods applied to specific safety regulations and domains such as automotive [69], avionics [58] and industrial control [55] domains. It is also a common industrial practice. The hardware and software support needed to facilitate deterministic timing analysis [70, 63] as well as probabilistic timing analysis [64] has also been object of deep analysis.

Note that quantitative comparisons across devices are not provided since WCET estimates are only partially dependent on the device used, its components and the nanoscale implementation aspects. Instead, WCET estimate trustworthiness and tightness strongly depend on how the device is configured for use (e.g., number of cores active, cache replacement policies used) and how the software is deployed – which limits potential interference, and, more importantly, on the particular timing analysis method used, where the cost to apply it, and the amount and type of input needed to apply it, vary dramatically across methods [67]. Overall, to the best of our knowledge, relevant quantitative comparisons across devices on the same ground (i.e. comparable configuration settings, same application, same timing analysis method) with realistic assumptions do not exist, and any conclusion for such a comparison could only be extremely dependent on the settings, applications and timing analysis method used, thus precluding from obtaining a general conclusion for devices compared. Instead, only deep – but qualitative – comparisons exist for few devices (e.g., [24]).

Software Scheduling Generally, safety standards demand simple, static and design-time defined scheduling solutions. For example, industrial safety standards such as IEC 61508-3 recommend the usage of design time deterministic scheduling methods (e.g., cycling scheduling with pre-assigned time slots) and strict priority scheduling by means of priority inversion avoidance [13, 4]. Hypervisors, such as XtratuM, support design time cyclic scheduling policies for partitions [59].

In order to improve the efficiency of resource sharing, and therefore the overall system average performance, several scheduling solutions have been proposed for mixed-criticality systems as summarized in the survey [75]. This is an active research topic in the scheduling research community that combines functions of different criticality to be scheduled with multi-core devices. However, as explained by Paulitsch et al. [74], with respect to industrial safety systems the contribution of this research is closer to the concepts of 'survivability' and 'graceful degradation' [76] rather than criticality and safety compliance (with respect to safety standards such as IEC 61508).

4 Domain Specific Approaches

This section extends the survey with other domains that do not use IEC 61508 as reference safety standard, avionics and aerospace domain. It also provides a summary of selected domain specific relevant example case-studies.

4.1 Avionics Domain

Avionic systems are subject to very strict certification processes that need to be approved by specific certification authorities. Thus, certification of safety-related systems based on multi-core devices in avionics is often more challenging than for other domains and, moreover, the avionic industry is even more reluctant than other industries to redesign already certified software.

The Integrated Modular Avionics (IMA) architecture, used by most avionic systems in the last decade, has enabled the integration of more functions in fewer computers, thus a reduction of computer systems SWaP. However, increasing performance demands in avionics (e.g., Airbus A-380 has 80 Mbytes code size [138]) cannot longer be satisfied with single core processors [164], and multi-core devices become mandatory to increase performance without increasing the number of on-board computers [164, 165, 17, 7, 10].

However, the multi-core integration (multi-IMA) of legacy single-core IMA systems is a certification challenge, because legacy partitions temporal and spatial isolation must be guaranteed without incurring in huge re-certification costs [165, 166, 167, 24, 168, 169, 170, 171]. The updated ARINC 653 supports the usage of multi-core devices but restricting the execution of safety partitions to a single core at a time with the remaining cores disabled, which is against SWaP reduction.

CAST-32A position paper [172], developed by avionics certification authorities, provides some guidance on how to use multi-core devices for safety-critical systems, allowing multiple cores to be used while running at least one safety partition, but there is still a significant gap between those guidelines and what COTS multi-core devices offer in terms of partition and control of interference channels. This gap relates to the abstract objectives provided in the CAST-32A position paper, which are hard to match – if at all possible – with the actual characteristics of COTS multi-core devices offering the performance level needed by avionic systems. Agirre et al. [173] analyze qualitatively and quantitatively such gap for the particular case of the NXP P4080 multi-core device, showing that COTS multi-core device design in general, and for the NXP P4080 device in particular, is at odds with the objectives imposed by CAST-32A position paper.

Efforts towards easing the adoption of multi-core devices in avionics safety-related systems have been abundant. Paulitsch et al. [74, 10] provide a research state-of-the-art description of mixed-criticality systems, with insights into real avionics systems. Some authors aim at characterizing the timing characteristics of shared hardware resources in COTS multi-core devices, either from a hardware perspective [170] or from an application perspective [174]. The required hypervisor support needed to limit interference due to contention in shared resources has also been evaluated and multiple alternatives provided [161, 175] as well as IMA compliant solutions for timing analysis of legacy code [176]. Agrawal et al. [177] provide a specific solution to manage memory bandwidth dynamically on COTS multi-core devices while still preserving a sufficient degree of time predictability. The impact of fault-tolerance mechanisms on time predictability for multi-core devices has also received some attention [58] since this aspect is too often overlooked. Finally, some authors consider the use of probabilistic timing analysis as a way of relieving end users from the cumbersome task of mastering execution time variability in complex multi-core devices [64, 178, 179].

4.2 Space Domain

Similarly to the avionics domain, the space domain has started building its systems on the IMA for Space (IMA-SP) architecture to facilitate the consolidation of multiple functions onto a single computer for SWaP reduction. However, as for the avionics domain, IMA-SP does not provide explicit solutions for the integration of multiple safety partitions on multi-core devices.

Space systems are subject to a number of specific requirements far different from those of terrestrial ones [180]. For example, legacy code is relatively scarce since each mission is basically unique and they require higher levels of fault tolerance support because they are generally exposed to much higher levels of radiation than terrestrial systems. However, as in many other safety-related domains, performance demands grow sustainedly, with code size scaling by a factor of 10x every 10 years, with missions in 2010 already reaching few millions of lines of code [180].

These characteristics impose the use of high-performance radiation-hardened hardware devices for spacecraft control and process data retrieved through instruments, as well as the consideration of approaches for system verification and validation such as timing aspects of multi-core devices. Such radiation-hardened devices, either in the form of ASIC processors or FPGA implementations, have been developed for several space missions (e.g., dual-core LEON3-FT [23]). In this context, multi-core devices are becoming dominant, with space industry considering increasingly parallel hardware devices such as the Next-Generation Microprocessor (NGMP) [153]. The landscape of future computing devices for space for on-board processing is analyzed in [45]. Apart from the aforementioned NGMP device, authors also identify the RAD5500 processor family, a radiation-hardened processor based on the e5500 core of the QorIQ Power Architecture processor. For instance, the RAD5545 processor employs four RAD5500 cores and uses 45 nm technology [45].

A detailed survey of the available (in 2012) space processors for space missions is given in [181], including radiation and non-radiation hardened processors, as well as those with no redundancy, dual-modular redundancy and triple-modular redundancy for fault tolerance. From a different perspective, other works aim at reconciling safety and security concerns for Integrated Modular Avionics for Space (IMA-SP) systems, but still without considering multi-core devices [182].

4.3 Example Case Studies

Table 3 provides a summary of selected domain specific relevant example case-studies based on a variety of multi-core devices and covering several fundamental safety technical requirements.

Domain	Device	Description
Automotive	AURIX (TC27x)	A safety concept example for ISO 26262 ASILD compliant automotive cruise-control safety system based on an automotive *safety device* and virtualization technology (hypervisor) [2]
	AURIX (TC2xx)	Powertrain control example based on a *safety device* using the proposed software development and test environment [183].
	Hercules (TMS570)	Example safety Anti-Lock Braking System (ABS) implementation in a single-core (TMS470) and dual-core *safety device* (TMS570). Both implementations are compared with respect to safety, task scheduling, real-time performance and power consumption [184].
	Generic	Simplified state of the art of automotive COTS multi-core devices, operating systems and timing analysis tools. It proposes a legacy code migration pattern for automotive multi-core devices taking into consideration associated safety standards [185].
Avionics	QorIQ (T4240)	Design and time analysis of a mixed-criticality system that integrates safety and non-safety software applications based on a *generic device* and virtualization technology (hypervisor) [7].
	QorIQ (P4080), ARM A15, Cyclone V, C6000	Selection criteria and assessment of multiple COTS devices against avionics requirements, together with suggestions for their appropriate use and modifications to EASA guidelines [24].
	QorIQ (P4080)	Qualitative and quantitative assessments of the use of this COTS multi-core device for avionic systems, with emphasis on timing analysis [170] and compliance against CAST-32A guidance [173].
	MPPA-256 Bostan	Assessment of software (PikeOS) and MPPA-256 device against avionics requirements in the context of multi-core devices [17].
	LEON3+	Part of the flight control system is evaluated on a 4-core LEON3 design implemented on a FPGA, together with ARINC 653 compliant PikeOS in the context of probabilistic timing analysis [186].
Railway	QorIQ (P4080)	A safety concept example for EN 5012x SIL4 compliant onboard railway signalling safety system based on a *generic device* and hypervisor technology [3].
Industrial Control	Core 2 Duo Processor	Industrial robot controller that integrates mixed-criticality functions such as real-time control and IEC 61508 SIL2 safety soft-PLC in a dual core processor [26].
	Intel Atom (x86) and LEON3FT	Safety certification strategy and safety concept examples for IEC 61508 SIL3 compliant wind-turbine safety system based on *hybrid safety devices* and hypervisor technology [54, 6, 4, 55].
	Zynq-7000 (ARM, MicroBlaze)	Safety concept for IEC 61508 SIL3 compliant wind-turbine safety system based on *generic device* and hypervisor technology [54, 6, 4], extended with certification strategy that supports variability, product line and modular safety cases [7].
Medical Systems	Zynq-7000	Real-time assessment of dependable health-care case study that integrates Linux-KVM with Memguard, on-chip communication and security services [7].

	Several	Overview of processor fault-tolerant techniques and solutions that target medical applications [5]

Table 3: Summary of selected example case-studies

5 Conclusion and Future Research Directions

This section describes links to other related research topics, overall conclusion and future research directions.

5.1 Links to other research topics

When several software applications of different criticality are integrated in a multi-core device, multiple challenges need to be addressed and managed, in order to support the cost-effective safety certification of the given integrated solution. This section explores the links between the surveyed research scope and some additional research challenges to be considered.

5.1.1 Parallel Application Software

The development of software applications for multi-core devices needs to consider the parallel / concurrent execution of multiple software applications on a single device with shared resources.

- Software development: Previously described hypervisor virtualization and operating system extensions that support partitioning provide a foundation for the integration of software partitions with parallel / concurrent execution [4].
- Software re-usability: The re-usability of sequential legacy code (mono-core) becomes a challenge because it needs to consider both the parallel execution of software applications and the usage of shared resources [42, 12, 187]. For example, Macher [12] describes generic steps that can be used to migrate legacy software to multi-core devices.
- Software integration: The integration of multiple mixed-criticality parallel application software partitions on multi-core devices is another field of research with the development of technical solutions (e.g., hypervisor and partitions) and strategies at system (e.g., [6]) and on-chip level (e.g., performance and run-time monitors [94]).

5.1.2 Parallel Programming Models

The definition of heterogeneous and parallel programming languages [188] with support for mixed-criticality safety systems is a challenge to be addressed [189], which could help exploit the performance delivered by multi-core devices enabling the execution of parallel software applications, rather than only multiple sequential applications in parallel. In this context, OpenMP has been regarded as a potentially appropriate solution, enabling functional verification, and being already supported by Keystone II and MPPA multi-core devices [189]. Regarding timing verification for real-time systems, two different OpenMP models have been considered so far in the literature, namely fork-join [190] and tasking [191] models, being the latter the one gaining more popularity due to its ability to materialize parallelism even at fine granularity, and not being restricted to only structured parallelism. Irregular task graphs can also be parallelized with the tasking model, and research initiatives aim at leveraging support for heterogeneous and asynchronous parallelism to deliver time predictability [192].

5.1.3 Heterogeneous Computing Platforms

In addition to multi-core devices, heterogeneous computing platforms including GPUs and application specific architectures (e.g., tensor cores) can potentially be used for the development of mixed-criticality systems. For example, GPUs provide high performance computing platforms for application domains such autonomous driving solutions [193, 194] (e.g., NVIDIA Xavier ASILC)). Mittal et. al [194] provide a survey of GPU based heterogeneous computing techniques and Alcaide et al. [193] an analysis and techniques for the usage of GPUs in high-integrity autonomous driving solutions with respect to ISO 26262 standard.

Besides, the development of *artificial intelligence* based autonomous systems is leading to open research problems such as the distribution of learning and inference activities over heterogeneous computing platforms. Upcoming SoC-FPGAs platforms (e.g., Xilinx Versal) combine these heterogeneous resources, but challenges remain with respect to hardware support for safety-critical systems such as predictable interconnects, avoidance of temporal interference in memory and safety monitors. For example, while the portability to different GPU architectures and programming interfaces was addressed in prior work [195], portability to other resource types and the simultaneous usage of heterogeneous computing resources is also considered a challenge, with few works currently addressing this challenge [196].

5.1.4 Product Line and Modular Certification Strategies

During the product lifetime different aspects needs to be considered in order to support complexity management [197] and support the incremental certification of systems [7], e.g., product line and variability management [7], periodic updates of hardware that require re-certification (e.g., 10-15 years in automotive [8]), modular safety cases and modular certification approaches [89, 96, 7].

5.1.5 Security / Cybersecurity

Safety and security requirements are common in multiple domains ("no safety without security") such as avionics [74, 10, 30], control systems [198], automotive [8] and railway [25, 30]. In addition to this, the industrial trend towards Industry 4.0 and digitalization, and the automotive trend towards autonomous vehicles, further exacerbates the need to comply with safety and security requirements. The definition of security standards compatible with safety standards development processes (e.g., IEC 62443 [199] and IEC 61508) and the definition of cross-domain verification and validation methods [30], paves the way towards safety and security systems certification. From a technical perspective this is an active field of research [198]. For example, the Multiple Independent Levels of Security / Safety (MILS) architectural approach combines safety and security requirements in different domains [74] and it is supported by several hypervisor solutions (e.g., XtratuM). In addition to this, several COTS *generic* and *safety* devices already provide support for the development of safe and secure applications.

Other important aspect is related to software updates. From a security point of view, it is essential to keep the system up-to-date with the latest security patches, what requires regular software updates. On the contrary, the service life of safety-critical systems is rather static, where operation time modifications involve a well established procedure that can not be trivially applied to frequent updates. Accordingly, a safety and security co-engineering is essential for the development and maintenance of such systems with *over-the-air software updates* [200].

5.1.6 Energy, Power and Thermal Management

As CMOS technology downscales, potentially more transistors can fit on the same silicon die area. However, power, thermal and energy-efficiency constraints limit this potential transistor density increase [37]. This design paradox is referred to as "dark silicon" [201]. In other to overcome these constraints, different approaches are considered such as thermally aware chiplets [202] and balance between fault-tolerance and power consumption [203]. Energy and power consumption is a critical factor for diverse safety-critical systems such as portable medical devices [5] and railway signalling autonomous remote object controllers [137]. In this context, Fakih et al. [137] describe techniques and approaches for energy, power and thermal management in the development of multi-core device based mixed-criticality systems, where described low power techniques must not jeopardize integrated safety functions.

5.1.7 Integrated Development Environments (IDEs) and tools

The development of multi-core device based mixed-criticality systems require the development or extension of IDEs and tools in order to support the complexity management derived from the integration of software functions of different criticality allocated to one or several cores (e.g., partition allocation) taking into consideration several technical constraints such as heterogeneous computing platforms, hypervisor configuration and partition scheduling, real-time guarantees, safety constraints, diagnosis, WCET, parallel programming models, product line variability and design space exploration [7]. For example, several research projects (e.g., ARAMiS II, EMC2, DREAMS) have addressed this challenge. A tool platform with common technical interfaces and seamless work-flows was defined in the ARAMiS II project, with a focus on development environments supporting partitioning, allocation, binding, scheduling and design space exploration for software components on multi-core devices [204]. Application models and design tools for multi-core device based mixed-criticality systems were defined in the EMC2 project [205]. For example, the developed tools support the optimized allocation of a system's functionality to a target hardware architecture, while ensuring bounded interference between different components in mixed-criticality settings. Development process and tools for mixed-criticality systems with networked multi-core devices were defined in the DREAMS project [7]. The tools support model-transformations, scheduling and design space exploration for mixed-criticality systems based on multi-core devices and virtualization technology.

5.2 Conclusions and Future Research Directions

The integration of functions of different safety criticality (e.g., safety and non-safety) in a multi-core device leads to safety certification challenges. This survey has categorized and summarized at different device abstraction levels (nanoscale, component and device) the state-of-the-art of selected key research contributions that support the compliance with fundamental safety technical requirements (reliability, diagnostic coverage, spatial and temporal independence). Most of the surveyed research state-of-the-art can be classified as either "how things should be done" (e.g., temporal independence with T-CREST time predictable device) or "how things can be done" (e.g., how to achieve temporal independence with current COTS *generic devices*). Taking into consideration the business oriented and innovative nature of the semiconductor industry and the required conservative nature of safety standards, in the future this divergent trend is expected to even increase.

With respect to safety certification, as described by the research state-of-the-art and example case-studies summarized in this survey, it is technically feasible to develop multi-core device based mixed-criticality safety systems. However, the techniques and strategies described in the case-studies have a high dependency with the integrated multi-core device and software applications (ad hoc solutions). Therefore, there is a need to consider cross-level and cross fundamental technical safety requirements research contributions. Although re-certification costs are also high, periodic system updates are commonly required due to hardware obsolescence, software updates and security patches. This leads to a need to manage either device stocks for the product life period [116] and / or techniques to decouple the dependency among the multi-core device and software applications, by means of techniques such as software partitioning and virtualization, safety and security co-engineering, safety product lines, modular safety cases and overall complexity management.

In addition to this, compliance with time independence requirement and supporting time predictability in multi-core devices is a technical and research challenge. The availability and development of novel architectures and components that provide time predictability (e.g., T-CREST) can simplify the WCET estimation and overall safety argumentation with respect to temporal independence. However, shared-resources based COTS multi-core devices are not generally designed with a focus on hard real-time applications and time predictability. Thus, there is a need for the definition of novel WCET analysis techniques, temporal diagnosis techniques and temporal independence strategies. In addition to this, parallel programming languages and parallel software application techniques that consider the parallel / concurrent execution of software are required for new software development, software re-usability and software integration.

Finally, as described by the International Technology Roadmap for Semiconductors (ITRS) [27], multi-core devices computational performance will continue increasing with further innovation and research in several fields such as DSM technologies, thermal management and device architectures. This will further require the development of novel hardware and software fault-tolerance, diagnosis techniques and strategies, which will also need to consider device energy-power-thermal management requirements and temporal predictability restrictions.

And this innovation is required for the development of next generation industrial and transportation systems that integrate autonomous systems, intelligent ADAS, machine learning and Industry 4.0 solutions, which will further accelerate the ever-increasing demand for additional functions to be integrated in a reduced number of computing nodes and processing devices with higher computational and on-chip communication performance. And this integration trend will further exacerbate current system integration engineering challenges such as off-chip communication capabilities, distribution of applications and services, distributed resource management, system level global-time and scheduling, IDEs and tools, system level safety and security engineering [7].

Acknowledgements

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness under grant TIN2015-65316-P, Basque Government under grant KK-2019-00035 and the HiPEAC Network of Excellence. The Spanish Ministry of Economy and Competitiveness has also partially supported Jaume Abella under Ramon y Cajal postdoctoral fellowship (RYC-2013-14717).

References

[1] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. In *IEEE Trans. on Dependable and Secure Comput.*, volume 1, pages 11–33, 2004.

[2] I. Agirre, M. Azkarate-askasua, A. Larrucea, J. Perez, T. Vardanega, and F. J. Cazorla. Automotive safety concept definition for mixed-criticality integration on a COTS multicore. In A. Skavhaug, J. Guiochet, E. Schoitsch, and F. Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 273–285. Springer Int. Publishing, 2016.

[3] I. Agirre, M. Azkarate-Askasua, A. Larrucea, J. Perez, T. Vardanega, and F. J. Cazorla. A safety concept for a railway mixed-criticality embedded system based on multicore partitioning. In *IEEE Int. Conf. on Comput. and Inform. Technol.; Ubiquitous Comput. and Commun.; Dependable, Autonomic and Secure Comput.; Pervasive Intell. and Comput. (CIT/IUCC/DASC/PICom)*, pages 1780–1787, 2015.

[4] J. Perez, D. Gonzalez, S. Trujillo, and A. Trapman. *A safety concept for an IEC 61508 compliant fail-safe wind power mixed-criticality embedded system based on multi-core partitioning*, volume 9111 of *Lecture Notes in Computer Science*. Springer Int. Publishing, 2015.

[5] M. Ottavi, D. Gizopoulos, and S. Pontarelli. *Dependable Multicore Architectures at Nanoscale*. Springer, 2018.

[6] J. Perez, D. Gonzalez, C. F. Nicolas, T. Trapman, and J. M. Garate. A safety certification strategy for IEC-61508 compliant industrial mixed-criticality systems based on multicore partitioning. In *17th Euromicro Conf. on Digital Syst. Design (DSD)*, pages 394–400, 2014.

[7] H. Ahmadian, R. Obermaisser, and J. Perez. *Distributed Real-Time Architecture for Mixed-Criticality Systems*. CRC Press, Taylor & Francis Incorporated, 2018.

[8] S. Chakraborty and S. Ramesh. Guest editorial special section on automotive embedded systems and software. *IEEE Trans. on Comput.-Aided Design of Integrated Circuits and Syst.*, 34(11):1701–1703, 2015.

[9] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz. From a federated to an integrated automotive architecture. *IEEE Trans. on Comput.-Aided Design of Integrated Circuits and Syst.*, 28(7):956–965, 2009.

[10] J. Athavale, R. Mariani, and M. Paulitsch. Flight safety certification implications for complex multi-core processor based avionics systems. In *IEEE 25th Int. Symp. on On-Line Testing and Robust Syst. Design (IOLTS)*, pages 38–39, 2019.

[11] D. Buttle. Real-time in the prime-time - ECRTS keynote talk. Technical report, ETAS Gmbh, 2012.

[12] G. Macher, A. Höller, E. Armengaud, and C. Kreiner. Automotive embedded software: Migration challenges to multi-core computing platforms. In *IEEE 13th Int. Conf. on Ind. Informatics (INDIN)*, pages 1386–1393, 2015.

[13] IEC. IEC 61508(-1/7): Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[14] ISO. ISO 26262(-1/11) road vehicles - functional safety, 2018.

[15] Claire Maiza, Hamza Rihani, Juan M. Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I. Davis. A survey of timing verification techniques for multi-core real-time systems. *ACM Comput. Surv.*, 52(3):56:1–56:38, 2019.

[16] J. Han, M. Deubzer, J. Seo Park, J. Harnisch, and P. Leteinturier. Efficient multi-core software design space exploration for hybrid control unit integration. In *SAE Tech. Paper*, 2014.

[17] S. Saidi, R. Ernst, S. Uhrig, H. Theiling, and B. D. de Dinechin. The shift to multicores in real-time and safety-critical systems. In *10th Int. Conf. on Hardware/Software Codesign and Syst. Synthesis*, pages 220–229. IEEE Press, 2015.

[18] G. Corradi. Tools, architectures and trends on industrial all programmable heterogeneous MPSoC (keynote). In *29th Euromicro Conf. on Real-Time Syst. (ECRTS)*, 2017.

[19] A. Hayek and J. Börcsök. Safety chips in light of the standard IEC 61508: Survey and analysis. In *Int. Symp. on Fundamentals of Electrical Eng. (ISFEE)*, pages 1–6, 2014.

[20] G. Blake, R. G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Mag.*, 26(6):26–37, 2009.

[21] Infineon. AURIX ^{TM} 32-bit microcontrollers for automotive and industrial applications. Report, Infineon, 2018.

[22] Xilinx. UG1085 - Zynq UltraScale+ device - technical reference manual. Report, Xilinx, 2018.

[23] V. Izosimov, A. Paschalis, P. Reviriego, and H. Manhaeve. *Application-Specific Solutions*. Springer, 2018.

[24] X. Jean, M. Gatti, G. Berthon, and M. Fumey. The use of multicore processors in airborne systems (EASA 2011.C31). Report, EASA, Thales Avionics, 2011.

[25] A. Bilbao, I. Yarza, J. L. Montero, M. Azkarate-askasua, and N. Gonzalez. A railway safety and security concept for low-power mixed-criticality systems. In *IEEE 15th Int. Conf. on Ind. Informatics (INDIN)*, pages 59–64, 2017.

[26] Case study, Intel core 2 duo processor on KUKA robot controller. `https://www.intel.com/content/dam/doc/case-study/industrial-core-kuka-study.pdf`, 2020.

[27] ITRS. International roadmap for devices and systems - executive summary. Report, IEEE, 2018.

[28] Meinhard Erben, Wolf Günther, Tobias Sedlmeier, Dieter Lederer, and Klaus-Jürgen Amsler. Legal aspects of safety designed software development, especially under european law. In *3rd Eur. Embedded Real Time Software (ERTS)*, page 6, 2006.

[29] EN. EN50128 - railway applications: Communication, signalling and processing systems - software for railway control and protection systems, 2011.

[30] Erwin Schoitsch. D6.1 - State of the art for system qualification and certification, V&V survey (EMC2). Report, 2013.

[31] R. Mariani, G. Boschi, and F. Colucci. Using an innovative SoC-level FMEA methodology to design in compliance with IEC 61508. In *Design, Automation and Test in Europe (DATE)*, pages 1–6, 2007.

[32] S. Kiamehr, M. B. Tahoori, and L. Anghel. *Manufacturing Threats*. Springer, 2018.

[33] C. Bolchini, M. K. Michael, A. Miele, and S. Neophytou. *Dependability Threats*. Springer, 2018.

[34] I. Schagaev and T. Kaegi-Trachsel. *Software Design for Resilient Computer Systems*. Springer Int. Publishing, 2016.

[35] D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera. Architectures for online error detection and recovery in multicore processors. In *Design, Automation and Test in Europe (DATE)*, pages 1–6, 2011.

[36] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *50th ACM/EDAC/IEEE Design Automation Conf. (DAC)*, pages 1–10, 2013.

[37] S. Das. *Variation-Mitigation for Reliable, Dependable and Energy-Efficient Future System Design*. Springer, 2018.

[38] V. Sridharan and S. Gurumurthi. *Resilience Proportionality - A Paradigm for Efficient and Reliable System Design*. Springer, 2018.

[39] R. Kalayappan and S. R. Sarangi. A survey of checker architectures. *ACM Comput. Surv.*, 45(4):1–34, 2013.

[40] H. Bauer, M. Patel, N. Santhanam, and B. Wiseman. Mckinsey on semiconductors. Report, 2015.

[41] World semiconductor trade statistics. `https://www.wsts.org/`, 2018.

[42] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theilingx. Multicore in real-time systems - temporal isolation challenges due to shared resources. In *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Syst. (WICERT)*, 2013.

[43] D. Dasari, B. Akesson, V. Nélis, M. A. Awan, and S. M. Petters. Identifying the sources of unpredictability in COTS-based multicore systems. In *8th IEEE Int. Symp. on Ind. Embedded Syst. (SIES)*, pages 39–48, 2013.

[44] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor system-on-chip (MPSoC) technology. *IEEE Trans. on Comput.-Aided Design of Integrated Circuits and Syst.*, 27(10):1701–1713, 2008.

[45] G. Furano and A. Menicucci. *Roadmap for On-Board Processing and Data Handling Systems in Space*. Springer, 2018.

[46] Ikhwan Lee, Michael Sullivan, Evgeni Krimer, Dong Wan Kim, Mehmet Basoglu, Doe Hyun Yoon, Larry Kaplan, and Mattan Erez. Survey of error and fault detection mechanisms. Report, The University of Texas, 2012.

[47] M. Ottavi, S. Pontarelli, D. Gizopoulos, C. Bolchini, M. K. Michael, L. Anghel, M. Tahoori, A. Paschalis, P. Reviriego, O. Bringmann, V. Izosimov, H. Manhaeve, C. Strydis, and S. Hamdioui. Dependable multicore architectures at nanoscale: The view from europe. *IEEE Design & Test*, 32(2):17–28, 2015.

[48] H. Lu. *Low-Cost Highly-Efficient Fault Tolerant Processor Design for Mitigating the Reliability Issues in Nanometric Technologies*. Thesis, 2013.

[49] Daniel J. Sorin. *Fault Tolerant Computer Architecture*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[50] C. Bernardeschi, L. Cassano, and A. Domenici. SRAM-based FPGA systems for safety-critical applications: A survey on design standards and proposed methodologies. *J. Comput. Sci. Technol.*, 30(2):373–390, 2015.

[51] T S Nidhin, Anindya Bhattacharyya, R P Behera, and T Jayanthi. A review on SEU mitigation techniques for FPGA configuration memory. *IETE Tech. Review*, 35(2):157–168, 2018.

[52] Jason A Cheatham, John M Emmert, and Stan Baumgart. A survey of fault tolerant methodologies for FPGAs. *ACM Trans. Des. Autom. Electron. Syst.*, 11(2):501–533, 2006.

[53] M. Paolieri, J. Mische, S. Metzlaff, M. Gerdes, E. Quiñones, S. Uhrig, T. Ungerer, and F. J. Cazorla. A hard real-time capable multi-core SMT processor. *ACM Trans. Embed. Comput. Syst.*, 12(3):79:1–79:26, 2013.

[54] S. Trujillo, A. Crespo, A. Alonso, and J. Perez. MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems. *Microprocessors and Microsystems*, 38(8, Part B):921 – 932, 2014.

[55] A. Larrucea, I. Agirre, C. F. Nicolas, J. Perez, M. Azkarate-Askasua, and T. Trapman. Temporal independence validation of an IEC-61508 compliant mixed-criticality system based on multicore partitioning. In *Forum on Specification and Design Languages (FDL)*, pages 1–8, 2015.

[56] I. Oz and S. Arslan. A survey on multithreading alternatives for soft error fault tolerance. *ACM Comput. Surv.*, 52(2):1–38, 2019.

[57] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Timing verification of fault-tolerant chips for safety-critical applications in harsh environments. *IEEE Micro*, 34(6):8–19, 2014.

[58] A. Löfwenmark and S. Nadjm-Tehrani. Fault and timing analysis in critical multi-core systems: A survey with an avionics perspective. *Journal of Syst. Architecture*, 87:1–11, 2018.

[59] A. Crespo, P. Balbastre, K. Chappuis, J. Coronel, J. Fanguède, P. Lucas, and J. Perez. *Execution Environment*. CRC Press, 2018.

[60] Z. Gu and Q. Zhao. A state-of-the-art survey on real-time issues in embedded systems virtualization. *Journal of Software Eng. and Appl.*, 5(1):277–290, 2012.

[61] Miloš Panić, Sebastian Kehr, Eduardo Quiñones, Bert Boddecker, Jaume Abella, and Francisco J. Cazorla. Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores. In *Int. Conf. on Hardware/Software Codesign and Syst. Synthesis (CODES)*, pages 29:1–29:10. ACM, 2014.

[62] Theo Ungerer, Christian Bradatsch, Martin Frieb, Florian Kluge, Jörg Mische, Alexander Stegmeier, Ralf Jahr, Mike Gerdes, Pavel Zaykov, Lucie Matusova, Zai Jian Jia Li, Zlatko Petrov, Bert Böddeker, Sebastian Kehr, Hans Regler, Andreas Hugl, Christine Rochange, Haluk Ozaktas, Hugues Cassé, Armelle Bonenfant, Pascal Sainrat, Nick Lay, David George, Ian Broster, Eduardo Quiñones, Milos Panic, Jaume Abella, Carles Hernandez, Francisco Cazorla, Sascha Uhrig, Mathias Rohde, and Arthur Pyka. Parallelizing industrial hard real-time applications for the parMERASA multicore. *ACM Trans. Embed. Comput. Syst.*, 15(3):53:1–53:27, 2016.

[63] H. Mushtaq, Z. Al-Ars, and K. Bertels. Calculation of worst-case execution time for multicore processors using deterministic execution. In *25th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 33–39, 2015.

[64] F. J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and T. Vardanega. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.*, 52(1):14:1–14:35, 2019.

[65] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstr. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.

[66] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla. Contention in multicore hardware shared resources: Understanding of the state of the art. In H. Falk, editor, *14th Int. Workshop on Worst-Case Execution Time Analysis*, pages 31–42, 2014.

[67] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Commy, M. Azkarate-Askasua, J. Perez, E. Mezzetti, and T. Vardanega. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE Int. Symp. on Ind. Embedded Syst. (SIES)*, 2015.

[68] P. Puschner. Embedded systems for safety-critical and mixed-criticality applications. In *2nd Mediterranean Conf. on Embedded Comput. (MECO)*, pages 1–2, 2013.

[69] K. Schmidt, J. Harnisch, D. Marx, A. Mayer, A. Kohn, and R. Deml. Timing analysis and tracing concepts for ECU development. *SAE World Congr. & Exhibition*, 1, 2014.

[70] S. Bensalem, K. Goossens, C. M. Kirsch, R. Obermaisser, E. A. Lee, and J. Sifakis. Time-predictable and composable architectures for dependable embedded systems. In *Ninth ACM Int. Conf. on Embedded Software (EMSOFT)*, pages 351–352, 2011.

[71] R. Zalman, A. Griessing, and P. Emberson. Timing correctness in safety-related automotive software. In *SAE Tech. Papers*. SAE Int., 2011.

[72] T. Mitra, J. Teich, and L. Thiele. Time-critical systems design: A survey. *IEEE Design & Test*, 35(2):8–26, 2018.

[73] Haluk Ozaktas, Christine Rochange, , and Pascal Sainrat. Automatic WCET analysis of real-time parallel applications. In *13th Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2013.

[74] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. Mixed-criticality embedded systems - a balance ensuring partitioning and performance. In *Euromicro Conf. on Digital Syst. Design (DSD)*, pages 453–461, 2015.

[75] A. Burns and R. I. David. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6), 2018.

[76] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. In *1st Int. Workshop on Mixed Criticality Syst. (WMC)*, pages 19–24, 2013.

[77] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Syst. Architecture*, 61(9):449–471, 2015.

[78] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. B. Nejad, A. Nelson, and S. Sinha. Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow. *SIGBED Rev.*, 10(3):23–34, 2013.

[79] C. Bradatsch, F. Kluge, and T. Ungerer. A cross-domain system architecture for embedded hard real-time many-core systems. In *IEEE 10th Int. Conf. on High Performance Comput. and Commun. (HPCC)*, pages 2034–2041, 2013.

[80] Carles Hernández, Jaume Abella, Francisco J. Cazorla, Alen Bardizbanyan, Jan Andersson, Fabrice Cros, and Franck Wartel. Design and Implementation of a Time Predictable Processor: Evaluation With a Space Case Study. In *29th Euromicro Conf. on Real-Time Syst. (ECRTS)*, volume 76, pages 16:1–16:23, 2017.

[81] D. Bui, E. Lee, I. Liu, H. Patel, and J. Reineke. Temporal isolation on multiprocessing architectures. In *48th ACM/EDAC/IEEE Design Automation Conf. (DAC)*, pages 274–279, 2011.

[82] K. Goossens and A. Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *47th Design Automation Conf. (DAC)*, pages 306–311. ACM, 2010.

[83] P. Puschner, B. Cilku, and D. Prokesch. Constructing time-predictable MPSoCs: Avoid conflicts in temporal control. In *IEEE 10th Int. Symp. on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 321–328, 2016.

[84] Arthur Pyka, Mathias Rohde, and Sascha Uhrig. A real-time capable coherent data cache for multicores. *Concurrency and Computation: Practice & Experience, Wiley*, 26(6):1342–1354, 2014.

[85] M. Schoeberl, W. Puffitsch, S. Hepp, B. Huber, and D. Prokesch. Patmos: a time-predictable microprocessor. *Real-Time Syst.*, 54(2):389–423, 2018.

[86] A. Platschek, N. Mc Guire, and L. Bulwahn. Certifying linux: Lessons learned in three years of SIL2LinuxMP. In *Embedded World*, 2018.

[87] I. Allende, N. Mc Guire, J. Perez, L. G. Monsalve, N. Uriarte, and Obermaisser R. Towards linux for the development of mixed-criticality embedded systems based on multi-core devices. In *15th Eur. Dependable Comput. Conf. (EDCC)*, pages 47–54, 2019.

[88] A. Larrucea, I. Martinez, H. Ahmadian, R. Obermaisser, V. Brocal, S. Peiró, and J. Perez. DREAMS: Cross-domain mixed-criticality patterns. In *Mixed-Criticality Workshop on Real Time System Symp. (RTSS)*, 2016.

[89] A. Larrucea, J. Perez, and R. Obermaisser. A modular safety case for an IEC 61508 compliant generic COTS processor. In *IEEE Int. Conf. on Comput. and Inform. Technol.; Ubiquitous Comput. and Commun.; Dependable, Autonomic and Secure Comput.; Pervasive Intell. and Comput. (CIT/IUCC/DASC/PICom)*, pages 1788–1795, 2015.

[90] S. Pinto, A. Oliveira, J. Pereira, J. Cabral, J. Monteiro, and A. Tavares. Lightweight multicore virtualization architecture exploiting ARM trustzone. In *43rd IEEE Ind. Electron. Society (IECON)*, pages 3562–3567, 2017.

[91] E. Grade, A. Hayek, and J. Börcsök. Implementation of a fault-tolerant system using safety-related Xilinx tools conforming to the standard IEC 61508. In *Int. Conf. on Syst. Rel. and Science (ICSRS)*, pages 78–83, 2016.

[92] A. Larrucea, J. Perez, and R. Obermaisser. A modular safety case for an IEC 61508 compliant COTS multi-core device. In *13th Int. Conf. on Dependable, Autonomatic and Secure Comput. (DASC)*, page 8, 2015.

[93] A. Hayek and J. Börcsök. SRAM-based FPGA design techniques for safety related systems conforming to IEC 61508 a survey and analysis. In *2nd Int. Conf. on Advances in Computational Tools for Eng. Appl. (ACTEA)*, pages 319–324, 2012.

[94] A. Crespo, P. Balbastre, J. Simó, J. Coronel, D. Gracia Pérez, and P. Bonnot. Hypervisor-based multicore feedback control of mixed-criticality systems. *IEEE Access*, 6:50627–50640, 2018.

[95] G. Nelissen, D. Pereira, and L. M. Pinho. A novel run-time monitoring architecture for safe and efficient inline monitoring. In Juan Antonio de la Puente and Tullio Vardanega, editors, *Reliable Software Technologies – Ada-Europe*, pages 66–82. Springer Int. Publishing, 2015.

[96] A. Larrucea, J. Perez, I. Agirre, V. Brocal, and R. Obermaisser. A modular safety case for an IEC 61508 compliant generic hypervisor. In *18th Euromicro Conf. on Digital Syst. Design (DSD)*, pages 571–574, 2015.

[97] A. Hayek, M. Schreiber, B. Machmur, and J. Börcsök. Design and implementation of on-chip safety controller in terms of the standard IEC 61508. In *Recent Advances in Circuits; Syst. and Automatic Control*, 2013.

[98] A. Hayek, B. Machmur, M. Schreiber, J. Börcsök, S. Gölz, and M. Epp. HICore1: Safety on a chip; turnkey solution for industrial control. In *IEEE 25th Int. Conf. on Application-Specific Syst., Architectures and Processors (ASAP)*, pages 74–75, 2014.

[99] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee. FlexPRET: A processor platform for mixed-criticality systems. In *IEEE 19th Real-Time and Embedded Technol. and Appl. Symp. (RTAS)*, pages 101–110, 2014.

[100] M. P. Zimmer. *Predictable Processors for Mixed-Criticality Systems and Precision-Timed I/O*. Thesis, 2015.

[101] I. Liu, J. Reineke, D. Broman, M. Zimmer, and E. A. Lee. A PRET microarchitecture implementation with repeatable timing and competitive performance. In *IEEE 30th Int. Conf. on Comput. Design (ICCD)*, pages 87–93, 2012.

[102] I. Liu, J. Reineke, and E. A. Lee. A PRET architecture supporting concurrent programs with composable timing properties. In *44th Asilomar Conf. on Signals, Syst. and Comput. (ASILOMAR)*, pages 2111–2115, 2010.

[103] M. Delvai, W. Huber, P. Puschner, and A. Steininger. Processor support for temporal predictability - the SPEAR design example. In *15th Euromicro Conf. on Real-Time Syst. (ECRTS)*, 2003.

[104] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory bandwidth management for efficient performance isolation in multi-core platforms. In *IEEE Trans. Comput.*, pages 562 – 576. IEEE, 2016.

[105] H. Yun. Improving real-time performance on multicore platforms using memguard, 2013.

[106] N. Dagieu, A. Spyridakis, and D. Raho. Memguard: A memory bandwith management in mixed criticality virtualized systems memguard KVM scheduling. In *10th Int. Conf. on Mobile Ubiquitous Comput., Syst., Services and Technologies (UBICOMM)*, 2016.

[107] L. Ecco, S. Tobuschat, S. Saidi, and R. Ernst. A mixed critical memory controller using bank privatization and fixed priority scheduling. In *IEEE 20th Int. Conf. on Embedded and Real-Time Comput. Syst. and Appl. (RTCSA)*, pages 1–10, 2014.

[108] J. Jalle, E. Quiñones, J. Abella, L. Fossati, M. Zulianello, and F. J. Cazorla. A dual-criticality memory controller (DCmc): Proposal and evaluation of a space case study. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 207–217, 2014.

[109] Y. Li, B. Akesson, and K. Goossens. Dynamic command scheduling for real-time memory controllers. In *26th Euromicro Conf. on Real-Time Syst. (ECRTS)*, pages 3–14, 2014.

[110] A. Barros and L. M. Pinho. Software transactional memory as a building block for parallel embedded real-time systems. In *37th Euromicro Conf. on Software Eng. and Advanced Appl. (SEAA)*, pages 251–255, 2011.

[111] C. Belwal and A. M. K. Cheng. Lazy versus eager conflict detection in software transactional memory: A real-time schedulability perspective. *IEEE Embedded Syst. Letters*, 3(1):37–41, 2011.

[112] M. El-Shambakey and B. Ravindran. STM concurrency control for embedded real-time software with tighter time bounds. In *49th Design Automation Conf. (DAC)*, pages 437–446. ACM, 2012.

[113] J. E. Gottschlich and D. A. Connors. Extending contention managers for user-defined priority-based transactions. In *Workshop on Exploiting Parallelism with Transactional Memory and other Hardware Assisted Methods (EPHAM)*, 2008.

[114] S. Metzlaff, S. Weis, and T. Ungerer. Leveraging transactional memory for a predictable execution of applications composed of hard real-time and best-effort tasks. In *21st Int. Conf. on Real-Time Networks and Syst. (RTNS)*, pages 45–54. ACM, 2013.

[115] S. Goossens, B. Akesson, and K. Goossens. Conservative open-page policy for mixed time-criticality memory controllers. In *Design, Automation and Test in Europe (DATE)*, pages 525–530, 2013.

[116] E. A. Lee. Cyber physical systems: Design challenges. In *11th IEEE Int. Symp. on Object Oriented Real-Time Distrib. Comput. (ISORC)*, pages 363–369, 2008.

[117] Giovani Gracioli, Ahmed Alhammad, Renato Mancuso, Antônio Augusto Fröhlich, and Rodolfo Pellizzoni. A survey on cache management mechanisms for real-time embedded systems. *ACM Comput. Surv.*, 48(2):32:1–32:36, 2015.

[118] P. K. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *IEEE Real-Time and Embedded Technol. and Appl. Symp. (RTAS)*, pages 1–12, 2016.

[119] A. Burns, J. Harbin, and L.S. Indrusiak. A wormhole NoC protocol for mixed criticality systems. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 184–195, 2014.

[120] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany. Survey on real-time networks-on-chip. *IEEE Trans. on Parallel and Distributed Syst.*, 28(5):1500–1517, 2017.

[121] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE Int. Symp. on Networks-on-Chip (NOCS)*, pages 161–170, 2008.

[122] C. Paukovits. *The time-triggered system-on-chip architecture.* Thesis, Institut für Technische Informatik, 2008.

[123] C. Iordanou, V. Soteriou, and K. Aisopos. Hermes: Architecting a top-performing fault-tolerant routing algorithm for networks-on-chips. In *2014 IEEE 32nd Int. Conf. on Comput. Design (ICCD)*, pages 424–431, 2014.

[124] C. Hilton and B. Nelson. PNoC: a flexible circuit-switched NoC for FPGA-based systems. *IEEE Proceedings - Comput. and Digital Techniques*, 153(3):181–188, 2006.

[125] A. Agarwal, L. Bao, J. Brown, and et al. The tile processor (TM) architecture: Embedded multicore for networking and digital multimedia. In *IEEE Hot Chips 19 Symp. (HCS)*, pages 1–12, 2007.

[126] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz. The time-triggered system-on-a-chip architecture. In *IEEE Int. Symp. on Ind. Electron. (ISIE)*, pages 1941–1947, 2008.

[127] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *IEEE Design & Test of Comput.*, 22(5):414–421, 2005.

[128] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Design, Automation and Test in Europe (DATE)*, volume 2, pages 890–895, 2004.

[129] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *J. Syst. Archit.*, 50(2-3):105–128, 2004.

[130] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.

[131] B. Anuradha and C. Vivekanandan. Usage of scratchpad memory in embedded systems — state of art. In *3rd Int. Conf. on Comput., Communication and Networking Technologies (ICCCNT)*, pages 1–5, 2012.

[132] A. Meixner, M. E. Bauer, and D. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. In *40th Annual IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pages 210–222, 2007.

[133] Shidhartha Das. *Razor: A variability-tolerant design methodology for low-power and robust computing.* Thesis, 2009.

[134] A. Larrucea, I. Martinez, R. Obermaisser, J. Perez, and C. F. Nicolas. Modular development of dependable mixed-criticality embedded systems. In *20th Euromicro Conf. on Digital Syst. Design (DSD)*, pages 419–426, 2017.

[135] S. Pontarelli, Juan A. Maestro, and P. Reviriego. *Dependability Solutions.* Springer, 2018.

[136] X. Iturbe, B. Venu, J. Jagst, E. Ozer, P. Harrod, C. Turner, and J. Penton. Addressing functional safety challenges in autonomous vehicles with the Arm TCLS architecture. *IEEE Design & Test*, 35(3):7–14, 2018.

[137] M. Fakih, A. Lenz, M. Azkarate-Askasua, J. Coronel, A. Crespo, S. Davidmann, J. C. Diaz Garcia, N. Romero Gonzalez, K. Grüttner, S. Schreiner, R. Seyyedi, R. Obermaisser, A. Maleki, J. Öberg, M. T. Mohammadat, J. Perez-Cerrolaza, I. Sander, and I. Söderquist. SAFEPOWER project: Architecture for safe and power-efficient mixed-criticality systems. *Microprocessors and Microsystems*, 52:89–105, 2017.

[138] J. Abella, F. J. Cazorla, E. Quiñones, A. Grasset, S. Yehia, P. Bonnot, D. Gizopoulos, R. Mariani, and G. Bernat. Towards improved survivability in safety-critical systems. In *IEEE 17th Int. On-Line Testing Symp. (IOLTS)*, pages 240–245, 2011.

[139] S. Guertin and M. White. CMOS reliability challenges - the future of commercial digital electronics and NASA. In *NEPP Electron. Technol. Workshop*, 2010.

[140] Xilinx. UG116 - device reliability report - first half 2018. Report, Xilinx, 2018.

[141] E. Hallet, G. Corradi, and S. McNeil. WP461 - Xilinx reduces risk and increases efficiency for IEC61508 and ISO26262 certified safety applications. Report, Xilinx, 2015.

[142] M. A. Alam, K. Roy, and C. Augustine. Reliability and process-variation aware design of integrated circuits - a broader perspective. In *Int. Rel. Physics Symp.*, pages 4A.1.1–4A.1.11, 2011.

[143] B. Keller, M. Cochet, B. Zimmer, J. Kwak, A. Puggelli, Y. Lee, M. Blagojević, S. Bailey, P. F. Chiu, P. Dabbelt, C. Schmidt, E. Alon, K. Asanović, and B. Nikolić. A RISC-V processor SoC with integrated power management at submicrosecond timescales in 28 nm FD-SOI. *IEEE Journal of Solid-State Circuits*, 52(7):1863–1875, 2017.

[144] A. Vajda. *Multi-core and Many-core Processor Architectures*, pages 9–43. Springer US, Boston, MA, 2011.

[145] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero. An analyzable memory controller for hard real-time CMPs. *IEEE Embedded Syst. Letters*, 1(4):86–90, 2009.

[146] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *19th IEEE Real-Time and Embedded Technol. and Appl. Symp. (RTAS)*, pages 55–64, 2013.

[147] G. Tsamis, S. Kavvadias, A. Papagrigoriou, M. D. Grammatikakis, and K. Papadimitriou. Efficient bandwidth regulation at memory controller for mixed criticality applications. In *11th Int. Symp. on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2016.

[148] Z. Owda and R. Obermaisser. Mixed-criticality transactional memory controller for embedded systems. pages 104–110, 2017.

[149] B. Krishna Priya, Amit D. Joshi, and N. Ramasubramanian. A survey on performance of on-chip cache for multi-core architectures. In *Int. Conf. on Informatics and Analytics (ICIA)*, pages 35:1–35:7. ACM, 2016.

[150] A. Scolari, F. Sironi, D. Sciuto, and M. D. Santambrogio. A survey on recent hardware and software-level cache management techniques. In *IEEE Int. Symp. on Parallel and Distrib. Processing with Appl.*, pages 242–247, 2014.

[151] S. Mittal. A Survey of Recent Prefetching Techniques for Processor Caches. *ACM Comput. Surv.*, 49(2), 2016.

[152] S. Mittal. A Survey of Techniques for Cache Partitioning in Multicore Processors. *ACM Comput. Surv.*, 50(2), 2017.

[153] Leonidas Kosmidis, Jerome Lachaize, Jaume Abella, Olivier Notebaert, Francisco J. Cazorla, and David Steenari. GPU4S: Embedded GPUs in space. In *22nd Euromicro Conf. on Digital Syst. Design (DSD)*, 2019.

[154] Pedro Benedicte, Carles Hernandez, Jaume Abella, and Francisco J. Cazorla. HWP: Hardware Support to Reconcile Cache Energy, Complexity, Performance and WCET Estimates in Multicore Real-Time Systems. In *30th Euromicro Conf. on Real-Time Syst. (ECRTS)*, pages 3:1–3:22, 2018.

[155] Saud Wasly. *Architecture design for distributed mixed-criticality systems based on multi-core chips*. Thesis, 2018.

[156] Salma Hesham, Jens Rettkowski, Diana Göhringer, and Mohamed A. Abd El Ghany. Survey on real-time network-on-chip architectures. In Kentaro Sano, Dimitrios Soudris, Michael Hübner, and Pedro C. Diniz, editors, *Applied Reconfigurable Comput.*, pages 191–202, Cham, 2015. Springer Int. Publishing.

[157] Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, and Francisco J. Cazorla. Increasing confidence on measurement-based contention bounds for real-time round-robin buses. In *52nd Design Automation Conf. (DAC)*, pages 125:1–125:6. ACM, 2015.

[158] D. Dasari and V. Nelis. An analysis of the impact of bus contention on the WCET in multicores. In *IEEE 14th Int. Conf. on High Performance Comput. and Commun. (HPCC)*, pages 1450–1457, 2012.

[159] J. Bin. *Controlling Execution Time Variability Using COTS for Safety Critical Systems*. Thesis, 2014.

[160] Paul S. Levy. WP495 - Using Zynq-7000 SoC IEC 61508 Artifacts to Achieve ISO 13849 Compliance. Report, Xilinx, 2017.

[161] X. Jean. *Hypervisor Control of COTS Multi-Cores Processors in Order to Enforce Determinism for Future Avionics Equipment*. PhD thesis, Telecom ParisTech, 2015.

[162] J. Perez, M. Coppola, M. Faugère, D. Gracia Perez, M. Grammatikakis, A. Larrucea Ortube, A. Mouzakitis, A. Papagrigoriou, P. Petrakis, V. Piperaki, I. Sarasola, and G. Tsamis. *Evaluation.* CRC Press, 2018.

[163] R. Pellizzoni, A. Schranzhofer, Chen Jian-Jia, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *Design, Automation and Test in Europe (DATE)*, pages 741–746, 2010.

[164] O. Sander, F. Bapp, L. Dieudonne, T. Sandmann, and J. Becker. The promised future of multi-core processors in avionics systems. *CEAS Aeronautical Journal*, 8(1):143–155, 2017.

[165] MULCORS - use of multicore processors in airborne systems (EASA project.2011/6). Report, EASA, 2012.

[166] EASA. Certification memorandum - software aspects of certification - EASA. Report, 2013.

[167] EASA. Development assurance of airborne electronic hardware, 2011.

[168] L. M. Kinnan. Use of multicore processors in avionics and its potential impact on implementation and certification. *SAE Tech. Papers*, 2009.

[169] P. Huyck. ARINC 653 and multi-core microprocessors - considerations and potential impacts. In *IEEE/AIAA 31st Digital Avionics Syst. Conf. (DASC)*, pages 6B41–6B47, 2012.

[170] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *9th Eur. Dependable Comput. Conf. (EDCC)*, pages 132–143, 2012.

[171] S. Fisher. Certifying applications in a multi-core environment: a new approach gains success. Report, SYSGO AG, 2013.

[172] CAST. Multi-core Processors - Position Paper CAST-32A. Report, 2016.

[173] I. Agirre, J. Abella, M. Azkarate-Askasua, and F. J. Cazorla. On the tailoring of CAST-32A certification guidance to real COTS multicore architectures. In *12th IEEE Int. Symp. on Ind. Embedded Syst. (SIES)*, pages 1–8, 2017.

[174] J. Bin, S. Girbal, D. Gracia Pérez, A. Grasset, and A. Merigot. Studying co-running avionic real-time applications on multi-core COTS architectures. In *Embedded Real Time Software and Syst. Conf.*, 2014.

[175] S. Girbal, X. Jean, J. Le Rhun, D. Gracia Pérez, and M. Gatti. Deterministic Platform Software for Hard Real-Time Systems Using Multi-Core COTS. In *IEEE/AIAA 32nd Digital Avionics Syst. Conf. (DASC)*. IEEE, 2013.

[176] S. R. Msirdi. *Multicore - Interference - Integration - Scheduling - Hard real-time software.* Thesis, 2017.

[177] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch. Contention-aware dynamic memory bandwidth isolation with predictability in COTS multicores: An avionics case study. In *29th Euromicro Conf. on Real-Time Syst. (ECRTS)*, 2017.

[178] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *8th IEEE Int. Symp. on Ind. Embedded Syst. (SIES)*, pages 241–248, 2013.

[179] F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovino, Z. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetta, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. Timing analysis of an avionics case study on complex hardware/software platforms. In *Design, Automation and Test in Europe (DATE)*, pages 397–402, 2015.

[180] J. Abella and F.J. Cazorla. Chapter 9 - harsh computing in the space domain. In Augusto Vega, Pradip Bose, and Alper Buyuktosunoglu, editors, *Rugged Embedded Syst.*, pages 267 – 293. Morgan Kaufmann, Boston, 2017.

[181] R. Ginosar. Survey of processors for space. In *Data Syst. In Aerospace (DASIA)*, 2012.

[182] J. Windsor, K. Eckstein, P. Mendham, and T. Pareaud. Time and space partitioning security components for spacecraft flight software. In *30th IEEE/AIAA Digital Avionics Syst. Conf. (DASC)*, pages 8A5–1–8A5–14, 2011.

[183] G. Macher, M. Bachinger, and M. Stolz. Embedded multi-core system for design of next generation powertrain control units. In *13th Eur. Dependable Comput. Conf. (EDCC)*, pages 66–72, 2017.

[184] S. K. Jena and M. B. Srinivas. On the suitability of multi-core processing for embedded automotive systems. In *Int. Conf. on Cyber-Enabled Distrib. Comput. and Knowledge Discovery (CyberC)*, pages 315–322, 2012.

[185] P. Leteinturier, S. Brewerton, and K. Scheibert. Multicore benefits & challenges for automotive applications. In *SAE Tech. Paper*, 2008.

[186] F. J. Cazorla, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-Askasua, F. Wartel, L. Cucu, F. Cros, G. Farrall, A. Gogonel, A. Gianarro, B. Triquet, C. Hernandez, C. Lo, C. Maxim, D. Morales, E. Quinones, E. Mezzetti, L. Kosmidis, I. Aguirre, M. Fernandez, M. Slijepcevic, P. Conmy, and W. Talaboulma. PROXIMA: Improving measurement-based timing analysis through randomisation and probabilistic analysis. In *Euromicro Conf. on Digital Syst. Design (DSD)*, pages 276–285, 2016.

[187] H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, 2005.

[188] J. Diaz, C. Muñoz Caro, and A. Niño. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Trans. on Parallel and Distrib. Syst.*, 23(8):1369–1386, 2012.

[189] S. Royuela, A. Duran, M. A. Serrano, E. Quiñones, and X. Martorell. *A Functional Safety OpenMP* for Critical Real-Time Embedded Systems*, book section 16. Springer Int. Publishing, 2017.

[190] K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *31st IEEE Real-Time Syst. Symp.*, pages 259–268, 2010.

[191] Maria A. Serrano, Sara Royuela, and Eduardo Quiñones. Towards an OpenMP specification for critical real-time systems. In *14th Int. Workshop on OpenMP (IWOMP)*, pages 143–159, 2018.

[192] Eduardo Quiñones, Marko Bertogna, Erez Hadad, Ana Juan Ferrer, Luca Chiantore, and Alfredo Reboa. Big data analytics for smart cities: The H2020 CLASS project. In *11th ACM Int. Syst. and Storage Conf. (SYSTOR)*, page 130. ACM, 2018.

[193] S. Alcaide, L. Kosmidis, C. Hernandez, and J. Abella. High-integrity GPU designs for critical real-time automotive systems. In *Design, Automation & Test (DATE)*, pages 824–829, 2019.

[194] Sparsh Mittal and Jeffrey S. Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4):69:1–69:35, 2015.

[195] Arya Mazaheri, Johannes Schulte, Matthew Moskewicz, Felix Wolf, and Ali Jannesari. Enhancing the programmability and performance portability of GPU tensor operations. In *25th Euro-Par Conf.*, volume 11725 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 2019.

[196] Roger Pujol, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Generating and Exploiting Deep Learning Variants to Increase Heterogeneous Resource Utilization in the NVIDIA Xavier. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[197] Hermann Kopetz. *Simplicity is Complex: Foundations of Cyber-Physical System Design*. Springer Int. Publishing, 2019.

[198] S. Kriaa, L. Pietre-Cambacedes, M. Bouissou, and Y. Halgand. A survey of approaches combining safety and security for industrial control systems. *Rel. Eng. and Syst. Safety*, 139:156–178, 2015.

[199] IEC. IEC 62443-4-1: Security for industrial automation and control systems - part 4-1: Secure product development lifecycle requirements, 2018.

[200] Imanol Mugarza, Jorge Parra, and Eduardo Jacob. Software updates in safety and security co-engineering. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 199–210, Cham, 2017. Springer Int. Publishing.

[201] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. *IEEE Micro*, 32(3):122–134, 2012.

[202] F. Eris, A. Joshi, A. B. Kahng, Y. Ma, S. Mojumder, and T. Zhang. Leveraging thermally-aware chiplet organization in 2.5D systems to reclaim dark silicon. In *Design, Automation and Test in Europe (DATE)*, pages 1441–1446, 2018.

[203] A. Maheshwari, W. Burleson, and R. Tessier. Trading off transient fault tolerance and power consumption in deep submicron (DSM) VLSI circuits. *IEEE Trans. on Very Large Scale Integration (VLSI) Syst.*, 12(3):299–311, 2004.

[204] Raphael Weber. ARAMIS II. Tool interoperability and exchange formats. final workshop, Stuttgart, 2019.

[205] W. Weber, A. Hoess, J. v. Deventer, F. Oppenheimer, R. Ernst, A. Kostrzewa, P. Doré, T. Goubier, H. Isakovic, N. Druml, E. Wuchner, D. Schneider, E. Schoitsch, E. Armengaud, T. Söderqvist, M. Traversone, S. Uhrig, J. C. Pérez-Cortés, S. Saez, J. Kuusela, M. v. Helvoort, X. Cai, B. Nordmoen, G. Y. Paulsen, H. P. Dahle, M. Geissel, J. Salecker, and P. Tummeltshammer. The EMC2 project on embedded microcontrollers: Technical progress after two years. In *Euromicro Conf. on Digital Syst. Design (DSD)*, pages 524–531, 2016.